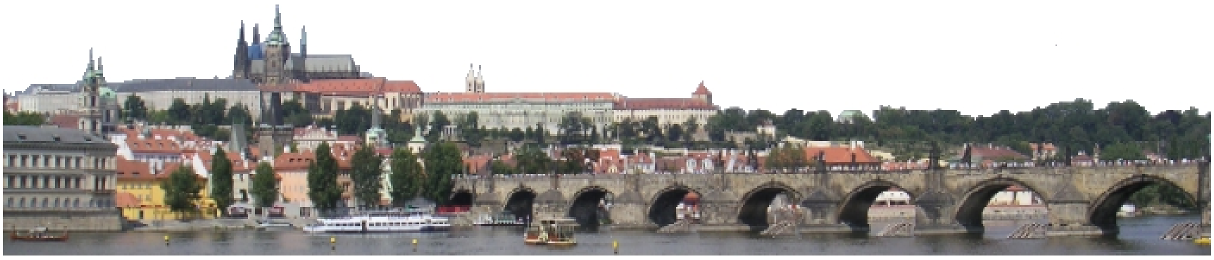
Proceedings of the 26th Workshop of the UK Planning and Scheduling Special Interest Group

PlanSIG 2007

December 17-18, 2007
Prague, Czech Republic

Editor: Roman Barták



Proceedings of PlanSIG 2007
The 26th workshop of the UK Planning and Scheduling Special Interest Group

December 17-18, 2007
Prague, Czech Republic

Workshop Chair

Roman Barták
Charles University in Prague
Faculty of Mathematics and Physics
Malostranské nám. 2/25, 118 00 Praha 1, Czech Republic
e-mail: bartak@ktiml.mff.cuni.cz

Programme Committee

Ruth Aylett, Herriot-Watt, UK
Chris Beck, University of Toronto, Canada
Ken Brown, University College Cork, Ireland
Edmund Burke, University of Nottingham, UK
Luis Castillo, University of Granada, Spain
Amedeo Cesta, ISTC, Italy
Alex Coddington, University of Strathclyde, UK
Stefan Edelkamp, Universität Dortmund, Germany
Susana Fernández, Universidad Carlos III de Madrid, Spain
Maria Fox, University of Strathclyde, UK
Antonio Garrido, Universidad Politecnica Valencia, Spain
Tim Grant, University of Pretoria, South Africa
Joerg Hoffmann, University of Innsbruck, Austria
Peter Jarvis, NASA Ames Research Center, USA
Graham Kendall, University of Nottingham, UK
Philippe Laborie, ILOG, France
John Levine, University of Strathclyde, UK
Derek Long, University of Strathclyde, UK
Lee McCluskey, University of Huddersfield, UK
Amnon Meisels, Ben-Gurion University, Israel
Barry O'Sulivan, University College Cork, Ireland
Sanja Petrovic, University of Nottingham, UK
Nicola Policella, European Space Agency, Germany
Julie Porteous, University of Strathclyde, UK
Patrick Prosser, University of Glasgow, UK
Hana Rudová, Masaryk University, Czech Republic
Wheeler Ruml, University of New Hampshire, USA
Rong Qu, University of Nottingham, UK
Sam Steel, University of Essex, UK
Andrew Tuson, City University, UK
Jozsef Vancza, SZTAKI, Hungary
Roman van der Krogt, 4C, Ireland
Petr Vilím, ILOG, France

ISSN 1368-5708

Table of Contents

SHORT PAPERS

Planning-based Scheduling for SLA-awareness and Grid Integration[*]

Dominic Battré and **Matthias Hovestadt** and **Odej Kao**
Technical University of Berlin, Germany
{battre,maho,okao}@cs.tu-berlin.de

Axel Keller and **Kerstin Voss**
Paderborn Center for Parallel Computing
University of Paderborn, Germany
{kel,kerstinv}@upb.de

Abstract

Service level agreements (SLAs) are powerful instruments for describing all obligations and expectations in a business relationship. It is of focal importance for deploying Grid technology to commercial applications. The EC-funded project HPC4U (Highly Predictable Clusters for Internet Grids) aimed at introducing SLA-awareness in local resource management systems, while the EC-funded project AssessGrid introduced the notion of risk, which is associated with every business contract. This paper highlights the concept of planning based resource management and describes the SLA-aware scheduler developed and used in these projects.

Introduction

In the academic domain Grid computing is well known, if not even established. Researchers are using Grid middleware systems like Unicore or the Globus Toolkit to create virtual organizations, dynamically sharing the transparent access to distributed resources. Grid computing started under the solely technical question of how to provide access to distributed high performance compute resources. Thanks to numberless projects and initiatives, funded by national and international bodies worldwide, Grid systems have significantly evolved meanwhile, making Grid technology adoptable in a large variety of usage scenarios.

Companies like IBM, Hewlett Packard, and Microsoft have recognized the potential of Grid Computing already in the early days of the Grid development, providing noticeable efforts on research and the support of research communities. However, the Grid did not really enter the commercial domain until the present day. Already in 2003 the European Commission (EC) convened a group of experts to clarify the demands of future Grid systems and which properties and capabilities are missing in current existing Grid infrastructures. Their work resulted in the idea of the Next Generation Grid (NGG) (Priol & Snelling 2003; Jeffery *(edt.)* 2004; De Roure *(edt.)* 2006). This work clearly identified that guaranteed provision of reliability, transparency, and Quality of Service (QoS) is an important demand for successfully commercialize future Grid systems. In particular, commercial users will not use a Grid system for computing business critical jobs, if it is operating on the best-effort approach only.

In this context, a Service Level Agreement (SLA) is a powerful instrument for describing all expectations and obligations in the business relationship between service consumer and service provider (Sahai *et al.* 2002). Such an SLA specifies the QoS requirement profile of a job. At the Grid middleware layer many research activities already focus on integrating SLA functionality.

The EC-funded project BeInGrid (Business Experiments in Grid (BeInGrid), EU-funded Project) aims at fostering the commercial uptake of the Grid. BeInGrid encompasses numerous business experiments, where Grid technology is to be introduced to specific business domains. Successful experiments reached the goal of proving the benefit of applying Grid technology for commercial customers. According to the NGG, a major objective in these BeInGrid experiments is the provision of reliability as contractually expressed in negotiated SLAs.

Current resource management systems (RMS) are working on the best-effort approach, not giving any guarantees on job completion to the user. Since these RMS are offering their resources to Grid systems, Grid middleware has only limited means in fulfilling all terms of negotiated SLAs.

For closing this gap between the requirements of SLA-enabled Grid middleware and the capabilities of RMS, HPC4U (Highly Predictable Cluster for Internet-Grids (HPC4U)) started working on an SLA-aware RMS, utilizing the mechanisms of process-, storage- and network-subsystems for realizing application-transparent fault toler-

[*]This work has been partially supported by the EU within the 6th Framework Programme under contract IST-031772 ”Advanced Risk Assessment and Management for Trustable Grids” (AssessGrid) and IST-511531 ”Highly Predictable Cluster for Internet-Grids” (HPC4U)

ance. As central component of the HPC4U project the RMS OpenCCS has been selected, since its planning based nature seemed to be well-suited for realizing SLA-awareness. Within the project all features required for SLA-awareness and SLA-compliance have been developed, e. g. an SLA-aware scheduler, mechanisms for transparent checkpointing of parallel applications, or the negotiation of new SLAs.

The HPC4U project will end 2007. The outcome of the project allows the Grid to negotiate on SLAs with the RMS. The RMS is only allowed to accept a new SLA, if it can ensure its fulfillment. For this, the RMS provides mechanisms like process and storage checkpointing to realize fault tolerance and to assure the adherence with given SLAs even in the case of resource failures. The HPC4U system is even able to act as an active Grid component, migrating checkpointed jobs to arbitrary Grid resources, if that allows the completion of the job according to its SLA.

In this paper we first highlight the concept of planning based resource management, a fundament for realizing SLA-aware RMS. The main part of the paper focuses on the specific demands of different job types on the scheduling as well as the scheduling impact of a Grid integration. The paper ends with an overview about related work and a short conclusion.

Planning Based Resource Management

Compute clusters have a long tradition beginning in the early 1970s with the UNIX operating system (Pfister 1997). Since then many resource management systems evolved, bringing functionality targeted to their specific usage domain, e. g. capabilities on load balancing. Classic systems are mostly used in capacity computing environments, computing large amounts of data in time uncritical context.

Most of the resource management systems available today can be classified as *queuing based systems*. The scheduler of these RMS is operating one or more queues, each of them with different priorities, properties, or constraints (e. g. high priority queue, weekend queue) (Windisch *et al.* 1996). Each incoming job request is assigned to one of these queues. The scheduling component of the RMS then orders each queue according to the strategy of the currently active scheduling policy. A very basic strategy is FCFS (First Come, First Served), assigning resources to jobs according to the job's entry time into the system. Modern RMS are also using priority queues, reflecting the status of the particular jobs. However, resources are assigned to jobs from the queue head, if the system has enough free resources. If this results in idle resources, backfilling strategies can be applied for selecting matching jobs from one of the queues for immediate out-of-order execution.

Many different strategies on backfilling have evolved, each optimizing according to a specific objective or usage environment. Commonly known strategies are conservative and EASY backfilling. Both strategies only differ in their way of selecting jobs for backfilling. While conservative backfilling demands that the backfilled job may not delay other waiting requests (Mu'alem & Feitelson 2001), EASY backfilling only demands the queue head's jobs not to be delayed (Lifka 1995). For deciding about the impact of a backfilling decision on the delay of jobs in the queues, the system has to have runtime information of these jobs. Hence, specific backfilling strategies (like EASY and conservative backfilling) can only be applied to environments where these statements are available.

By switching the focus from classic high throughput computing to computation of deadline bound and business critical jobs, also the demand on the RMS and its scheduler component changes. If negotiating on service level agreements, the system has to know about future utilization, i. e. whether it is possible to agree on finishing the new job as requested.

Planning is an alternative approach on system scheduling (Hovestadt *et al.* 2003). In contrast to queuing, planning does not only regard currently free resources and assigns them to waiting jobs. Instead, planning based systems also plan for the future, assigning a start time to all waiting requests. This way a schedule is generated, encompassing all jobs in the schedule. Having such a schedule available, the system scheduler is able to determine which jobs are scheduled to be executed at what time. Table 1 depicts the most significant differences between queuing and planning based systems.

A prerequisite for planning based resource management system is the availability of run time estimates for all jobs. Without this information the scheduler has no means to decide how long a specific resource will be used by a job. Hence, the scheduler could not assign a start time to jobs following in the schedule. In case the user underestimated the runtime, the system can try to extend the runtime of this job. If this is not possible because other jobs are scheduled on the resource, having a high priority so that they cannot be pushed away, the job has to be terminated or suspended in order to have the resources available for other jobs. This may be considered as a drawback of planning based resource management. A further drawback regards the cost of scheduling. The scheduling process itself is significantly more complex than in queuing based systems.

The novel approach on scheduling in planning based resource management systems allows the development of new scheduling policies and paradigms. Beside the classic policies like FCFS, SJF (Shortest Job First), or LJF (Longest Job First), novel policies could help to realize new objectives or new functionalities. We are convinced that planning based resource management is a good starting point for realizing SLA-awareness.

Scheduling for Typical Scenarios

In this section typical scenarios will be described. Starting with the submission of a regular local job, the degree of service quality will increase with each scenario. For realizing SLA-awareness in the EC-funded projects HPC4U and AssessGrid, the resource management system OpenCCS has been used. OpenCCS is a planning based resource management system developed at the University of Paderborn. Details on OpenCCS can be found in (Keller & Reinefeld 2001).

	queuing system	**planning system**
planned time frame	present	present and future
reception of new request	insert in queues	replanning
start time known	no	all requests
runtime estimates	not necessary[1]	mandatory
reservations	difficult	yes, trivial
backfilling	optional	yes, implicit
examples	PBS, NQE/NQS, LL	CCS, Maui Scheduler[2]

[1] exception: backfilling

[2] Maui may be configured to operate like a planning system (Jackson, Snell, & Clement 2001)

Table 1: Differences of queuing and planning systems (Hovestadt *et al.* 2003)

Local Job Submission

The local job submission is the classic case of job submission, where a user connects locally to the resource management system and submits a new job. Since OpenCCS is planning based, it requires all users to specify the expected duration of their requests. The OpenCCS planner distinguishes between *Fix-Time* and *Var-Time* resource requests. A *Fix-Time* request reserves resources for a given time interval. It cannot be shifted on the time axis. In contrast, *Var-Time* requests can move on the time axis to an earlier or later time slot (depending on the used policy). Such a shift on the time axis might occur when other requests terminate before the specified estimated duration.

The Planning Manager (PM) is a central component of the OpenCCS architecture, responsible for computing a valid, machine-independent schedule. Likewise, the Machine Manager (MM) is responsible for machine-dependent scheduling. The separation between the hardware independent PM and the system specific MM allows to encapsulate system specific mapping heuristics in separate modules. With this approach, system specific requests (e. g. for I/O-nodes, specific partition topologies, or memory constraints) may be considered. One task of the MM is to verify if a schedule received from the PM can be realized with the available hardware. The MM checks this by mapping the user given specification with the static (e. g. topology) and dynamic (e. g. PE availability) information on the system resources. Since OpenCCS is a planning-based RMS, the PM generates a schedule for both current and future resource usage. Therewith it supports classic scheduling strategies like FCFS, SJF, and LJF, considering aspects like project limits or system wide node limits. The system administrator can change the strategy during runtime.

The PM manages two lists while computing a schedule, which are sorted according to the active policy.

- The *New list(N-list)*: Each incoming request is placed in this list and waits there until the next planning phase begins.
- The *Planning list(P-list)*: These jobs have already been accepted by the system. The PM takes jobs from this list to generate the system schedule.

The PM first checks if the N-list has to be sorted according to the active policy (e. g. SJF or LJF). It then plans all elements of N-list. Depending on the request type (*Fix-Time* or *Var-Time*) the PM calls an associated planning function. For example, if planning a *Var-Time* request, the PM tries to place the request as soon as possible. The PM starts in the present and moves to the future until it finds a suitable place in the schedule.

Figure 1 depicts a typical schedule situation in a planning-based RMS. If a user submits a new job request, the system is able to match the request properties with the current schedule, i. e. the PM and MM components of OpenCCS are checking whether it is possible to generate a new valid system schedule. In this case, the user's job request is accepted, directly returning the time when the job will be allocated at the latest. If the request cannot be realized (e. g. because the user requested for a time slot with insufficient available resources), the job is rejected. In this situation, the user can query the system for the earliest possible time to start the job request.

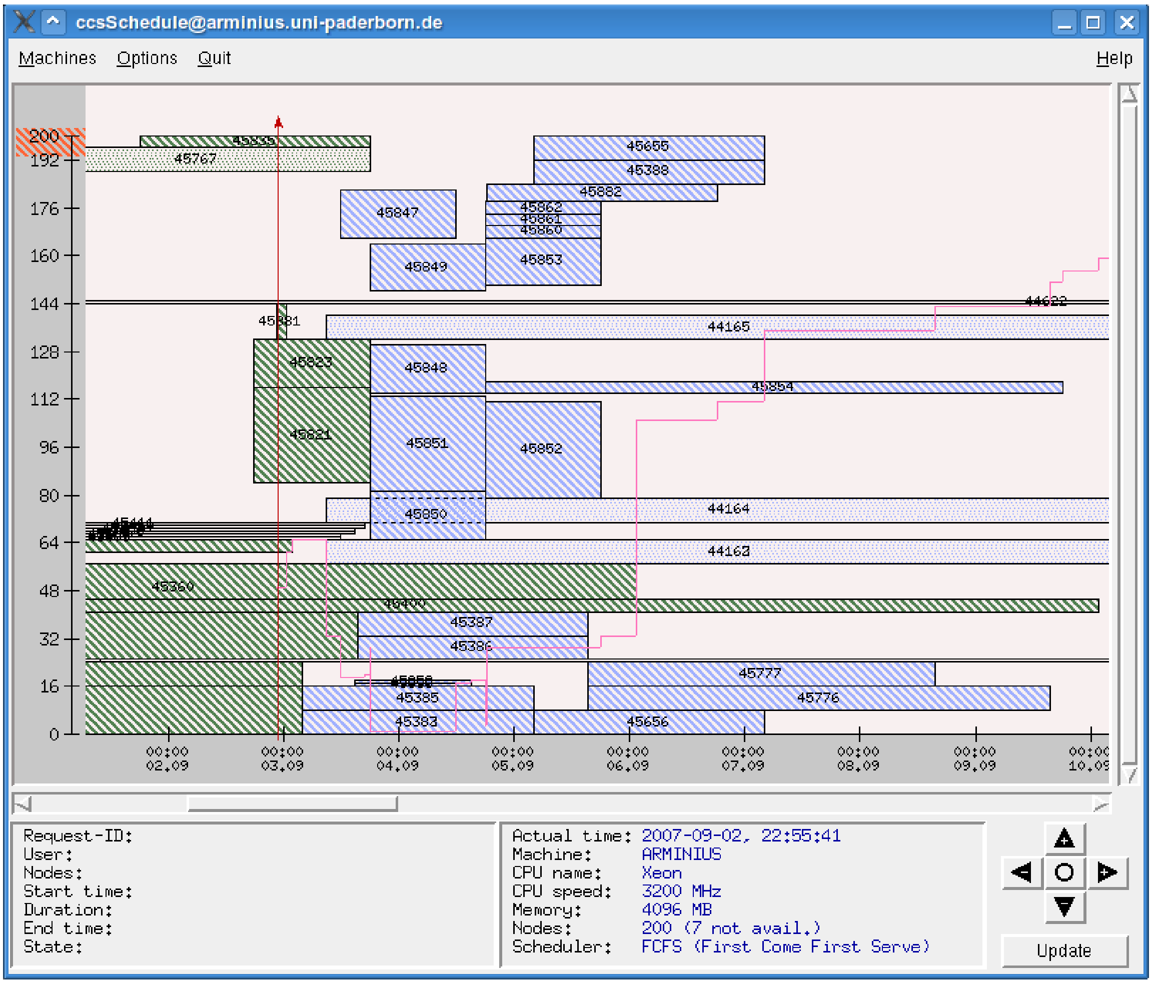
Deadline bound Jobs

Deadline bound jobs have to be completed until a specific time at the latest. A classic example for such a deadline bound job is a weather service which has to complete the computation of a weather forecast until 5am, since the forecast is to be broadcasted on TV at 6am. However, deadlines are also of particular importance for executing workflows, where the workflow is executed in multiple branches in parallel and where the result needs to be joined until a given time, so that also the overall workflow result can be delivered in time.

From the resource management system's point of view, a deadline bound job is a *Var-Time* resource requests. The user has to provide three key parameters:

- the number of required resources
- the duration of job execution
- the deadline for job completion

The deadline bound job is a specific case of a *Var-Time* resource request, since it may not shift arbitrarily on the time axis, but only within the boundaries given by the earliest possible start time and by the deadline. This constraint has to be regarded during the scheduling process, assigning resources early enough to allow the job to complete in time.



Request-ID:
User:
Nodes:
Start time:
Duration:
End time:
State:

Actual time: 2007-09-02, 22:55:41
Machine: ARMINIUS
CPU name: Xeon
CPU speed: 3200 MHz
Memory: 4096 MB
Nodes: 200 (7 not avail.)
Scheduler: FCFS (First Come First Serve)

Figure 1: Schedule in a planning based RMS

At this, the latest time for resource allocation conforms to the specified deadline minus the user's specified runtime.

In the case of deadline bound jobs, the correctness of the estimated runtime of the job is crucial for the fulfillment of the deadline. It is in the responsibility of the user to give a correct estimate. If the provider assigns a resource at the latest possible start time, it is the user's responsibility if the job did not complete in time, because he underestimated the job's runtime. However, users tend to overestimate the runtime of their jobs to prevent such a situation. Hence in the typical situation the job ends long before the estimated (and scheduled) end of time. Generally assuming the specified runtime to be overestimated allows to postpone the point of latest ressource allocation by the assumed amount of overestimation. However, this strategy is risky since jobs with correctly estimated runtimes will not be able to finish until their deadline.

Due to the nature of deadline bound jobs, the scheduler has to place them after placing all *Fix-Time* resource requests, but before placing regular *Var-Time* resource requests. At this, it follows the main scheduling policy, e. g. FCFS. The scheduler executes the following steps on an initially empty schedule, trying to place *Var-Time* resource requests at the earliest possible place in the new schedule:

1. sort all requests according to the current policy
2. place all *Fix-Time* resource requests (from first P-list, then from N-list)
3. place all deadline bound *Var-Time* resource requests (first from P-list, then from N-list)
4. place all remaining *Var-Time* resource requests (first from P-list, then from N-list)

Placing deadline bound *Var-Time* jobs according to policies like FCFS does not always result in a good schedule quality. Placing jobs in front of the schedule just because they arrived at the system at an early point of time (i. e. blocking valuable resources with this job) prevents executing other jobs with perhaps even nearer deadlines. Hence, other strategies could be applied when placing these deadline bound requests.

As an alternative approach, Deadline Monotonic Scheduling (DMS) (Audsley 1993) could be applied here, where the

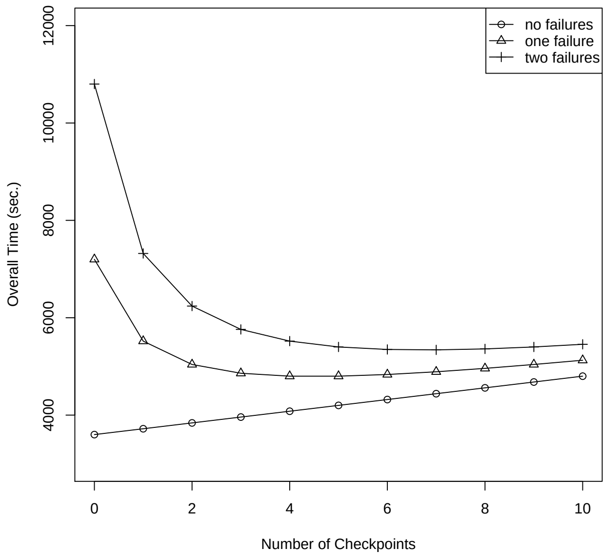


Figure 2: Impact of Checkpoint Frequency on Runtime

priority increases the nearer it gets to its deadline, i.e. the latest possible start time here. By applying Earliest Deadline First (EDF) (Buttazzo & Stankovic 1993), the scheduler would sort all deadline bound jobs by increasing remaining time until their latest possible point of start. This ensures that valuable resources are first used for urgent jobs.

Resource Failures and Fault Tolerance

A cluster system consists of multiple nodes. Partitions of these nodes are assigned to running applications, so that multiple applications are executed in parallel. If one of the nodes of a partition fails (e. g. due to a power outage), the execution of the application running on this node typically is aborted. In case of parallel applications, not only the processes of the application running on the affected node are aborted, but the entire parallel application is affected.

Cluster systems are used for speeding up the execution time of complex problems, but with an increasing grade of parallelism and an increasing runtime of the job (due to the complexity of the problem), also the possibility of a job crash increases, because only one of the nodes has to fail during the execution. This is a real problem for jobs running on dozens or hundreds of nodes over multiple days or weeks.

In the EC-funded project HPC4U mechanisms have been developed for transparently checkpointing parallel applications, i. e. all mechanisms can be applied without any modification of the job or relinking of the binary, even without having the job owner to take any notice of the mechanisms at all. This mechanism requires a patch to be applied to the Linux kernel, so that the process itself then runs inside a virtual bubble. At checkpoint time, the entire bubble is saved. For parallel applications, also the MPI implementation has to be enhanced, so that a consistent image of all parallel instances can be generated. For this purpose, the cooperative checkpoint protocol (CCP) has been developed.

Beside this stack of tools the project also evaluated other existing checkpointing solutions. At this, fairly good experiences also have been made with the tools Berkeley Checkpointing and Restart (BLCR) and LAM-MPI. Even if parallel checkpointing is possible, these tools have significant functionality drawbacks compared to the HPC4U stack.

By periodically checkpointing an application, the job can be restarted from the latest checkpointed state. Hence, only the computation steps after the latest checkpoint has to be repeated, instead of restarting the job from scratch. Even if the mechanisms have negligible impact on the job execution performance and the checkpointing of large jobs can be executed in a few seconds or minutes, this has to be considered at scheduling time.

Firstly, the effort for performing checkpoints enters the computation for the latest possible point of start. Since the time increases with the number of nodes and the amount of used memory, the system can predict quite exactly the time required for each checkpoint operation. The number of checkpoints determines the maximum time that can be lost due to a resource outage. It is a trade-off between reducing the worst-case loss of computational results and reducing the overhead of checkpointing.

The impact of the chosen checkpoint frequency on the runtime of a job is depicted in Figure 2. It assumes a job having a total runtime of one hour and a duration of each checkpoint of two minutes. The three curves represent the number of assumed resource outages. The curve depicting the case of no resource outages occurring has its minimum in $n = 0$, having no checkpoints generated. Since each checkpoint generation delays the completion of the job, each generated checkpoint is unnecessary overhead in the case of no resource outages. If no resource outages are expected or if a job restart is acceptable (like for best effort jobs), the best option is to execute without checkpoints.

In the case of resource outages occurring, things look different. An increasing number of checkpoints decreases the amount of lost compute steps lost through a resource outage, since the system is able to resume from the latest checkpointed state. The curves have their minima at the point of optimal trade-off between lost computation power and additional effort for executing the checkpoint operation. Moreover this number increases on increasing the number of expected outages. Where it is optimal to generate approximately four checkpoints in the case of one expected outage, it is approximately 7 in the case of two outages.

Secondly, the scheduling policy has to be adopted for handling the case of failures. If a job is affected by a resource outage, the entire job (not only the part of the failed node) is removed from the schedule. It leaves the P-list and is added to the *Defect list(D-list)*, encompassing all jobs affected by failures.

Then the scheduler starts the computation of a new system schedule, following the policies described above, placing jobs from D-list after jobs from P-list, but before placing jobs from N-list. This impacts new jobs (which may be rejected now), but does not impact other already planned jobs. However, if applying policies like DMS, the time until the job's latest point of start has to be recomputed, not taking the originally user specified job runtime into account, but the remaining runtime at the time of the last checkpoint.

The impact of resource failures on the system schedule can be reduced by introducing a failure horizon. A resource management system uses its internal monitoring mechanisms to detect problems within the cluster as soon as possible. If such a problem can not be solved by internal recovery mechanisms of the RMS itself, the cluster administrators are informed. The failure horizon represents the typical time required by administrators to solve such reported errors (e. g. 12 hours). The RMS only moves those jobs to the D-list which are planned on the defect resources within the failure horizon, assuming that the resource is available again at allocation time of all other jobs.

SLA Negotiation

The process of SLA-negotiation differs significantly from the regular job submission interface of a resource management system. There, a user submits his job description, directly getting an information about rejection or acceptance in return. In the latter case, the job has already entered the system schedule.

In case of service level agreements, a multi-phase negotiation is conducted before the job finally enters the system. The GRAAP working group (MacLaren 2003) of the Open Grid Forum (OGF) (Open Grid Forum) described such a negotiation process in the WS-Agreement Negotiation specification (Andrieux *et al.* 2004). Here the provider answers a job request with an SLA offer. The user has to commit to this offer before the SLA is actually enforced.

For the scheduling component of an RMS this negotiation process has significant implications: once the RMS has issued an SLA offer, it has to adhere to this offer until it has been committed or canceled by the user. Timeout mechanisms ensure that SLA offers automatically expire after a given time period (e. g. some seconds). However, at least during this timeout period the system has to reserve system capacity for the job in negotiation.

For this purpose, a novel list is introduced into the system: the *SLA-offer list(O-list)*. Jobs from this list are scheduled within the regular scheduling process in the order P-list before D-list before O-list before N-list. It is preferable to privilege jobs from D-list than O-list, since jobs in O-list are not yet affirmative, so that the system would not actually break an SLA-contract but only an SLA-offer. Again, the general policy of handling failures is to not affect other jobs, to keep the implication of a failure as local as possible. This also implies, that given SLA-offers should be kept if possible.

Data Staging of Grid Jobs

A second significant difference between locally submitted jobs and jobs coming from the Grid is the aspect of data staging. In case of local jobs it can be assumed that all necessary job data (e. g. the application binary and all input data) are available on a local computer system, so that fast local network connections can be used for transferring the data to the compute cluster. The time necessary for this can be neglected in general. In case of Grid jobs, this so called *stage-in* process has to be executed using slow WAN connections.

For this reason, the Grid user does not only have to specify parameters like estimated runtime, number of nodes, or deadline in the negotiation process, but also the earliest time for starting. The deadline can only be met if both the computation and the stage-in can be completed until this time. Since providers are usually connected over high bandwidth connections to the Internet, the bottleneck usually is the Internet network connection of the customer. Knowing the total amount of data that needs to be staged-in, he has to estimate the time required for transferring it over the Internet. The earliest point for starting the job is the time where the SLA has been committed (i. e. when the stage-in process could start) plus the total transfer time.

As long as the schedule has sufficient free space, the job may directly start after the estimated duration of the stage-in process. Overestimating the time for stage-in is uncritical, because this would only result in having the data available at RMS side earlier than expected. In contrast, if the user underestimated the stage-in time, the RMS is unable to start the job at the planned time. This directly threatens the fulfillment of the deadline, if the runtime is estimated correctly and there is no buffer between the planned end of the job and the deadline. The RMS has two options to handle such a situation, differing significantly in their demands on system management:

1. keeping the partition available for the job, waiting the start until stage-in is completed
2. assigning other waiting jobs to the pending job's resources, executing the pending job as soon as stage-in is completed

The first option does not require any specific RMS mechanisms, since the nodes of the pending job's partition simply remain idle. As soon as the stage-in process has been completed, the RMS starts the job. Even if this option is simple and easily manageable, it has two major disadvantages. First, the job is in danger of not finishing until the planned end, since the allocation time (i. e. the estimated runtime) is running while nodes are idle. Secondly, the overall cluster utilization is impacted, because nodes run idle instead of computing jobs.

The second option solves both of these problems, since nodes are used for computing other jobs and allocation time only starts when stage-in is completed. However, this option demands the system to support preemption of jobs. For this, we again use the checkpointing mechanisms developed in the HPC4U project. Since this solution provides transparent checkpointing for parallel applications, we are able to realize preemption for parallel jobs. For preempting a job, the job is first checkpointed and then stopped.

If other jobs are started in the partition of the pending job, these jobs have to be preempted. The scheduler is now able to rebuild the schedule after:

- subtracting the already executed runtime of the preempted jobs from their estimated runtime.

- setting the end of node allocation to the minimum of specified deadline and current time plus estimated job runtime.

This way, the job would have its entire estimated runtime available, as long as the delay in stage-in is not larger than the original buffer between end of computation and deadline. It has to be noted, that the deadline compliance of the preempted jobs is not endangered, because they already executed the time that they now get started later.

Accepting or Rejecting New Job Requests

In the previous sections it has been outlined how the demands on scheduling and system management increase with demands coming from deadline support or Grid interface. However, the general procedure of accepting or rejecting new job requests remains the same.

If a resource request is submitted to the RMS, the scheduler tries to build a new valid schedule that contains this new request. In case the scheduler succeeds, e. g. if the deadline of the new job can be realized without violating any other *Fix-Time* resource request or deadline bound *Var-Time* request, the new request is accepted by the system.

Related Work

The worldwide research in Grid computing resulted in numerous different Grid packages. Beside many commodity Grid systems, general purpose toolkits exist such as Unicore (UNICORE Forum e.V.) or Globus (Globus Alliance: Globus Toolkit). Although Globus represents the de-facto standard for Grid toolkits, all these systems have proprietary designs and interfaces. To ensure future interoperability of Grid systems as well as the opportunity to customize installations, the OGSA (Open Grid Services Architecture) working group within the OGF aims to develop the architecture for an open Grid infrastructure (GGF Open Grid Services Architecture Working Group (OGSA WG) 2003).

In (Jeffery *(edt.)* 2004), important requirements for the Next Generation Grid (NGG) were described. Among those needs, one of the major goals is to support resource-sharing in virtual organizations all over the world. Thus attracting commercial users to use the Grid, to develop Grid enabled applications, and to offer their resources in the Grid. Mandatory prerequisites are flexibility, transparency, reliability, and the application of SLAs to guarantee a negotiated QoS level.

An architecture that supports the co-allocation of multiple resource types, such as processors and network bandwidth, was presented in (Foster *et al.* 1999). The Globus Architecture for Reservation and Allocation (GARA) provides "wrapper" functions to enhance a local RMS not capable of supporting advance reservations with this functionality. This is an important step towards an integrated QoS aware resource management. In our paper, this approach is enhanced by SLA and monitoring facilities. These enhancements are needed in order to guarantee the compliance with all accepted SLAs. This means, it has to be ensured that the system works as expected at any time, not only at the time a reservation is made. The GARA component of Globus currently does neither support the definition of SLAs or malleable reservations, nor does it support resilience mechanisms to handle resource outages or failures.

The requirements and procedures of a protocol for negotiating SLAs were described in SNAP (Czajkowski *et al.* 2002). However, the important issue of how to map, implement, and assure those SLAs during the whole lifetime of a request on the RMS layer remains to be solved. This issue is also addressed by the architecture presented in this paper.

The Grid community has identified the need for a standard for SLA description and negotiation. This led to the development of WS-Agreement/-Negotiation (Andrieux *et al.* 2004).

Conclusion and Future Work

Introducing SLA-awareness is a mandatory prerequisite for the commercial update of the Grid. Consequently SLA-awareness also has to be introduced to local resource management systems which are currently operating on a best-effort approach. The EC-funded project HCP4U aims at providing an application-transparent and software-only solution of such an SLA-aware RMS, demanding for reliability and fault tolerance. The HPC4U system already allows the Grid user to negotiate on new SLAs, which will be realized by means like process-, network,- and storage-checkpointing.

In this paper we have described the requirements of various job types and their demands on an SLA-aware scheduling. In particular we addressed the implications of a Grid integration on the scheduling policies. The described scheduling rules have been implemented within the OpenCCS resource management system, which is used in the HPC4U project. Benefiting from the mechanisms of checkpointing and restart, the scheduler has proved to be well suited for executing jobs to their negotiated SLAs. Presuming that spare resources are not allocated by other SLA bound jobs, the system is able to cope with resource outages, fulfilling the SLAs of all jobs. Thanks to the transparent checkpointing capabilities, these mechanisms also apply for the execution of commercial applications, where no source code is available and recompiling or relinking is not possible. The user even does not have to modify the way of executing the job in the Grid. Hence, HPC4U reached its goal of providing transparent fault tolerance.

However, the availability of spare resources proved to be the limiting factor at restart time. If all resources of the cluster system are allocated by SLA bound jobs, the system has no means of restarting the failure affected job, thus violating the terms of its SLA.

Improving this situation is subject of currently ongoing work. Firstly, the notion of buffer nodes is introduced to the SLA-aware scheduler. These buffer nodes may only be used for executing best-effort jobs, so that outages either affect these buffer nodes or running best-effort jobs can be displaced by SLA-bound jobs that are affected by the resource outage. Secondly, the checkpoint and restart mechanisms will be used for suspending the execution of running jobs with respect to their SLA, thus freeing allocated resources for restarting outage affected jobs. Thirdly, the scheduler will actively select jobs for migration over the Grid, so that they can be finished on remote resources according to their SLA.

The scheduler is also the fundament for work done in the EC-funded project AssessGrid. Here, the notion of risk awareness and risk management is introduced into all layers of the Grid. This implies that the scheduler of the RMS has to consider risks of SLA violations in all scheduling decisions.

References

Andrieux, A.; Czajkowski, K.; Dan, A.; Keahey, K.; Ludwig, H.; Nakata, T.; Pruyne, J.; Rofrano, J.; Tuecke, S.; and Xu, M. 2004. Web Services Agreement Specification (WS-Agreement). http://www.gridforum.org/Meetings/GGF11/Documents/draft-ggf-graap-agreement.pdf.

Audsley, N. 1993. Deadline monotonic scheduling theory and application. *Control Engineering Practice* 1:71–78.

Business Experiments in Grid (BeInGrid), EU-funded Project. http://www.beingrid.eu.

Buttazzo, G. C., and Stankovic, J. 1993. Red: A robust earliest deadline scheduling algorithm. In *3rd intl. workshop on responsive computing systems*.

Czajkowski, K.; Foster, I.; Kesselman, C.; Sander, V.; and Tuecke, S. 2002. SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. In D.G. Feitelson, L. Rudolph, U. S. E., ed., *Job Scheduling Strategies for Parallel Processing, 8th InternationalWorkshop, Edinburgh,*.

De Roure *(edt.)*, D. 2006. Future for European Grids: GRIDs and Service Oriented Knowledge Utilities. Technical report, Expert Group Report for the European Commission, Brussel.

Foster, I.; Kesselman, C.; Lee, C.; Lindell, B.; Nahrstedt, K.; and Roy, A. 1999. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *7th International Workshop on Quality of Service (IWQoS), London, UK*.

GGF Open Grid Services Architecture Working Group (OGSA WG). 2003. Open Grid Services Architecture: A Roadmap.

Globus Alliance: Globus Toolkit. http://www.globus.org.

Highly Predictable Cluster for Internet-Grids (HPC4U), EU-funded project IST-511531. http://www.hpc4u.org.

Hovestadt, M.; Kao, O.; Keller, A.; and Streit, A. 2003. Scheduling in HPC Resource Management Systems: Queuing vs. Planning. In *Job Scheduling Strategies for Parallel Processing: 9th International Workshop, JSSPP, Seattle, WA, USA*.

Jackson, D.; Snell, Q.; and Clement, M. 2001. Core Algorithms of the Maui Scheduler. In D. G. Feitelson and L. Rudolph., ed., *Proceddings of 7th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, 87–103. Springer Verlag.

Jeffery *(edt.)*, K. 2004. Next Generation Grids 2: Requirements and Options for European Grids Research 2005-2010 and Beyond. ftp://ftp.cordis.lu/pub/ist/docs/ngg2_eg_final.pdf.

Keller, A., and Reinefeld, A. 2001. Anatomy of a resource management system for hpc clusters. *Annual Review of Scalable Computing* 3:1–31.

Lifka, D. A. 1995. The ANL/IBM SP Scheduling System. In D. G. Feitelson and L. Rudolph., ed., *Proc. of 1st Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, 295–303. Springer Verlag.

MacLaren, J. 2003. Advanced Reservations - State of the Art. Technical report, GRAAP Working Group, Global Grid Forum, http://www.fz-juelich.de/zam/RD/coop/ggf/graap/sched-graap-2.0.html.

Mu'alem, A., and Feitelson, D. G. 2001. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. In *IEEE Trans. Parallel & Distributed Systems 12(6)*, 529–543.

Open Grid Forum. http://www.ogf.org.

Pfister, G. 1997. *In Search of Clusters*. Prentice Hall.

Priol, T., and Snelling, D. 2003. Next Generation Grids: European Grids Research 2005-2010. ftp://ftp.cordis.lu/pub/ist/docs/ngg_eg_final.pdf.

Sahai, A.; Graupner, S.; Machiraju, V.; and van Moorsel, A. 2002. Specifying and Monitoring Guarantees in Commercial Grids through SLA. Technical Report HPL-2002-324, Internet Systems and Storage Laboratory, HP Laboratories Palo Alto.

UNICORE Forum e.V. http://www.unicore.org.

Windisch, K.; Lo, V.; Feitelson, D.; and Nitzberg, B. 1996. A Comparison of Workload Traces from Two Production Parallel Machines. In *6th Symposium Frontiers Massively Parallel Computing*, 319–326.

An Enhanced Weighted Graph Model for Examination/Course Timetabling

Julie R. Carrington[*], Nam Pham[†], Rong Qu[†], Jay Yellen[*†]

[*]Department of Mathematics and Computer Science, Rollins College
Winter Park, Florida, USA
{jcarrington, jyellen}@rollins.edu

[†]Automated Scheduling, Optimisation and Planning Research Group
School of Computer Science, University of Nottingham, Nottingham, UK
{nxp, rxq, jzy} @ cs.nott.ac.uk

Abstract

We introduce an enhanced weighted graph model whose vertices and edges have several attributes that make it adaptable to a variety of examination and course timetabling scenarios. In addition, some new vertex- and colour-selection heuristics arise naturally from this model, and our implementation allows for the use and manipulation of various combinations of them along with or separate from the classical heuristics that have been used for decades. We include a brief description of some preliminary results for our current implementation and discuss the further development and testing of the ideas introduced here.

Introduction

Background

Using graph colouring to model timetabling problems has a long history (e.g., Broder 1964, Welsh and Powell 1967, Wood 1968, Neufeld and Tartar 1974, Brelaz 1979, Mehta 1981, and Krarup and de Werra 1982). Several survey papers have been written on this topic (e.g., Schmidt and Strohlein 1980, de Werra 1985, Carter 1986, Schaerf 1999, Burke, Kingston, and deWerra 2004, and Qu et al. 2006).

In a standard graph representation for a timetabling problem, the events to be scheduled are represented by vertices. A constraint (conflict) between two events indicating that they should be assigned different time slots is represented by an edge between the two corresponding vertices. In our case, the events are exams (or courses) and the constraints might be that some students are enrolled in both exams or the same professor is giving both courses. Ideally, then, such exams (courses) would be assigned different time slots. If we associate each possible time slot with a different colour, then creating a conflict-free timetable is equivalent to constructing a *feasible* (or *proper* or *valid*) colouring of the vertices of the graph, that is, a vertex colouring such that *adjacent* vertices (two vertices joined by an edge) are assigned different colours.

Given that vertex colouring is *NP-Hard* (Papadimitriou and Steiglitz 1982), the development of heuristics and corresponding *approximate algorithms*, which forfeit the guarantee of optimality, has been a central part of the research effort.

Two events with a constraint between them are generally prohibited from being assigned the same time slot, i.e., the edge represents a *hard constraint*. In some university timetabling scenarios, another objective is to minimize the number of students that have to take exams close together (or courses far apart). This *proximity* restriction is generally regarded as a *soft constraint*.

The weighted graph model introduced in 1992 (Kiaer and Yellen 1992a) was designed to handle timetabling instances for which the number of available time slots (colours) is smaller than the minimum needed to construct a feasible colouring. (This minimum number is called the *chromatic number* of the graph.) For instance, in course timetabling, there is likely to be a limited number of time slots that can be used during the week, and a conflict-free timetable may not exist. If conflicts are unavoidable, then a choice must be made on which ones to accept.

Distinguishing among conflicts

Clearly, certain conflicts are worse than others. If two exams (or courses) require the same professor to be present or use the same equipment that cannot be shared, then those two exams must not be scheduled at the same time. On the other hand, if two exams happen to have one student in common, then scheduling those two exams in the same time slot may need to be considered acceptable. In fact, there may be situations where the distinction between hard and soft constraints becomes less clear. For instance, a timetable having a single student scheduled to take two exams in the same time slot (forcing some special accommodation) may actually be preferred to one that has 50 students taking back-to-back exams.

Scope of Paper

This paper introduces an extension of the weighted graph model of Kiaer and Yellen (1992a). This enhanced model holds and keeps track of more of the information relevant to the two sometimes opposing objectives – minimizing total conflict penalty (or keeping it zero) and minimizing total proximity penalty. A natural byproduct of this approach is the emergence of some new heuristics that appear to hold promise for their use, separately or in combination, in fast, one-pass, approximate algorithms.

Such algorithms can prove useful in a number of ways. Because solutions are produced quickly, they can be used within a flexible, interactive decision-support system that can be adapted to a variety of timetabling scenarios.

These solutions can also be used as initial solutions in local search and improvement based techniques, (e.g., Tabu Search, Simulated Annealing, Large Neighborhood Search, Case-Based Reasoning), or as upper bounds for a branch-and-bound algorithm (Kiaer and Yellen 1992b). Recent research has demonstrated that these algorithms, when hybridized effectively or integrated with other techniques such as meta-heuristics, are highly effective on solving timetabling problems (Qu et al. 2006).

Also, because the model lends itself to using various combinations of heuristics for vertex and colour selection, it may prove useful in the context of *hyper-heuristics* (Burke et al. 2003) and/or in an evolutionary computation approach that might involve automatic generation of combinations and switching from one combination to another as the colouring progresses (see Burke et al. 2007).

For an up-to-date survey that includes a broad overview and extensive bibliography of the research in this area in the last ten years (see Qu et al. 2006).

Description of the Model

Although we restrict our attention for this paper to examination timetabling, our model is also applicable to course timetabling. Moreover, it incorporates more of the problem information at input and keeps track of more information pertaining to the partial colouring during the colouring process than do existing timetabling models. These features led us to the design of some new vertex- and colour-selection heuristics, which we introduce in this paper.

Each vertex in the graph corresponds to an exam to be scheduled and each colour corresponds to a different time slot. Accordingly, assigning colour *c* to vertex *v* is taken to mean that the exam corresponding to *v* is scheduled in the time slot corresponding to *c*.

We represent various components of a typical instance of an Examination Timetabling problem using a weighted graph model. Each vertex and each edge are *weighted* with several attributes, some that hold information from the problem instance and others that hold and update information that helps guide the colouring process.

Associated with each vertex is the set of students who must take that exam. Two vertices are joined by an edge, and are said to be *adjacent* or *neighbors*, if it is undesirable to schedule the corresponding exams in the same time slot. Each edge carries information that tells us how undesirable it would be for the corresponding exams to be scheduled in the same time slot or in time slots near each other. In particular, each edge has two attributes: the set of students taking both exams (*intersection subset*); and a positive integer indicating the conflict severity if the exams are scheduled in the same time slot. This second attribute is currently tied to the size of the intersection subset. However, it can also reflect factors not tied to this intersection. For instance, if the same professor is assigned to both exams, then the severity is likely to be set at a high level.

To illustrate our model, suppose there are four available time slots, 0, 1, 2, and 3 for five exams, E1, E2, E3, E4, and E5. The set of students taking each of the exams is as follows:

E1: {a, b, …, j}
E2: {k, l, …, z}
E3: {a, e, k}
E4: {b, c, d, x, y, z}
E5: {a, c, e, g, i, j}

Each edge in the graph shown in Figure 1 has the subset of students enrolled in both exams corresponding to the endpoints of that edge.

In general it may be undesirable to assign the same time slot (colour) to a given pair of exams for a variety of reasons. For this example, however, we consider two vertices to be adjacent only if there is at least one student taking both exams.

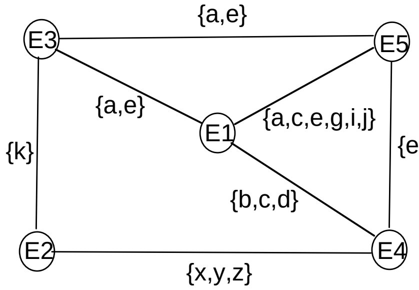


Figure 1: Student intersections for pairs of exams.

For our example, we set the conflict severity equal to 1, 5, or 25, according to the size of the intersection. In particular, we set the conflict severity to 1 if the intersection size is 1 or 2, to 5 if the intersection size is 3 or 4, and to 25 if the intersection size is 5 or greater (see Figure 2). We emphasize that these thresholds for conflict severity are arbitrarily chosen here. If a conflict-free timetable is a requirement, as it is in the University of Toronto problem instances (Carter, Laporte, and Lee 1996), then all conflict severities can simply be set to one since all conflicts are regarded as equally bad.

Of course, as mentioned, there will be many situations in which the conflict severity depends on other factors. In these situations, an edge might exist even when it corresponds to an empty intersection of students.

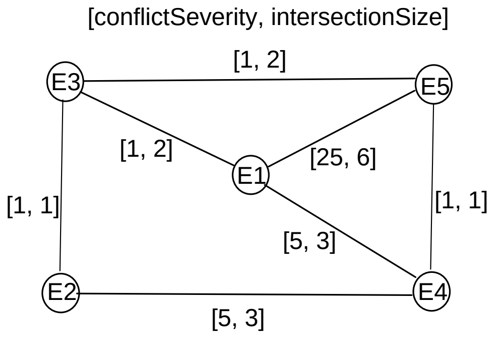


Figure 2: Additional edge attributes.

The proximity penalty of assigning colours c_i and c_j to the endpoints of an edge is a function of how close c_i and c_j are and the size of the intersection. For the Toronto problem instances, where the time slots are simply $c_i = i$, i = 0,1,…, the intersection size is multiplied by a *proximity weight* that equals $2^{5-|i-j|}$ when $|i-j| \leq 5$ and 0, otherwise. Our implementation uses this same evaluation for comparison purposes with the Toronto benchmark results. However, if the time slots are specified by a day, a start time, and a duration, then our colour attributes can easily be modified to allow for the appropriate change in the proximity evaluation function.

Our overall objective is to produce colourings (timetables) with minimum total conflict (zero may be required) and minimum total proximity penalty.

Knowing the conflict severity and size of the intersection for each edge makes it straightforward to keep track of the two kinds of penalties as the colouring progresses. When a vertex gets coloured *c*, that colour becomes less desirable (or forbidden) to its neighbors, as do colours in proximity with colour *c*.

Our model keeps track of these two kinds of colour undesirability as follows. Each vertex *v* has a *colour-penalties vector* that indicates the undesirability of assigning each colour to that vertex with respect to conflict penalty and proximity penalty. That is, the component of the colour penalties vector corresponding to colour *c* has two values, one is the conflict penalty incurred if *v* is coloured *c*, and the other is the resulting proximity penalty.

Using our example and a simplified proximity function, we illustrate how the colour-penalties vectors change as the graph is coloured. Suppose that any two colours *i* and *j* of the colours 0, 1, 2, and 3 are *within proximity* if they differ by 1, then the proximity penalty incurred when the colours of the endpoints of an edge differ by 1 equals the intersection size. Suppose further that the colour-penalties vectors for all of the vertices are initialized with [0, 0] for all of their colour components. Figure 3 shows the result of colouring vertex E1 with colour 1.

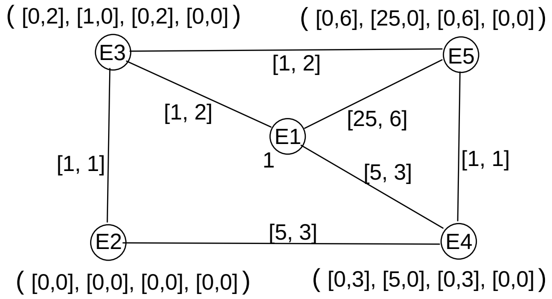


Figure 3: Colour-penalties vectors after E1 is coloured 1.

There may be other factors that make certain time slots undesirable for an individual exam. For instance, if professor X is assigned to exam A and cannot be on campus before noon. So any colour corresponding to a morning time slot for exam A would be given a prohibitively large conflict penalty value before the colouring begins.

As each vertex is coloured, its adjacent vertices' colour-penalties vectors are updated. The ease with which we are able to keep track of both hard and soft constraints as the colouring progresses creates new opportunities for the use of more sophisticated heuristics tied to this readily accessible information.

The Basic Approximate Algorithm

Our basic algorithm consists of two steps, select a vertex and then colour that vertex. We repeat these two steps until all vertices are coloured. Notice that while our model will easily accommodate more computation-intensive algorithms involving backtracking, local improvement, etc., we chose for this first phase of our research to concentrate on producing fast, essentially one-pass colourings.

Summary of the Model Features and Parameters

In preparation for the next section's discussion of heuristics, we list the key features and parameters on which the heuristics are based. The two edge attributes, conflict severity and intersection size, give rise to two different versions of the traditional concept of weighted degree of a vertex.

- *Conflict severity* (of an edge) – a measure of how undesirable it is to assign the same colour to both endpoints of the edge. In general, this would depend on several factors, and it could be set interactively by the end-user.
- *Intersection size* (of an edge) – the size of the intersection of the two sets corresponding to the endpoints of the edge. In exam timetabling, this is simply the number of students taking both exams.
- *Conflict degree* (of a vertex) – the sum of the conflict severities of the edges incident on the vertex.

- *Intersect degree* (of a vertex) – the sum of the intersection sizes of the edges incident on the vertex.
- *Bad-conflict edge* – an edge whose conflict severity exceeds a specified *threshold* value. If a conflict-free timetable (i.e., a feasible colouring) is required, then this threshold is set to zero, as we do for the Toronto problem instances.
- *Bad-intersect edge* – an edge whose intersection size exceeds a specified threshold. In our current implementation, this threshold is a function of the average of the intersection sizes of all edges; specifically, we use the average intersection size times some constant multiplier.
- *Conflict penalty* (for the colour assignment of a vertex) – a measure of how undesirable it is to assign that colour to the vertex. This will depend on the colour assignments of the vertex's neighbors and the conflict severities of the relevant edges, but it could also depend on other factors (e.g., professor, room, or equipment constraints).
- *Proximity* (of two colours) – a measure of how close together (in the case of exam timetabling) or spread apart (for course timetabling) the two colours are. This is often a secondary objective to optimize in school timetabling and is typically referred to as a *soft constraint*.
- *Proximity penalty* (for the colour assignment of a vertex) – the sum of the proximity penalties resulting from that colour assignment and the colour assignments of all neighbors of that vertex (determined by the function described immediately following Figure 2).
- *Colour-penalties vector* (of a vertex) – indicates for each colour the conflict penalty and proximity penalty of assigning that colour to the vertex. When a vertex is coloured, the colour-penalties vector of each of that vertex's neighbors must be updated accordingly.
- *Bad-conflict colour* (for a vertex) – a colour whose conflict penalty for that vertex exceeds some specified threshold (also set to zero for the Toronto instances since feasible colourings are required).
- *Bad-proximity colour* (for a vertex) – a colour whose proximity penalty for that vertex exceeds some specified threshold. Similar to the bad-intersect-edge threshold, we use average intersection size times a (possibly different) constant multiplier.

The thresholds for badness are easily adaptable to the requirements of the problem, and, in a decision support system, they could be specified by the end-user interactively. Part of this ongoing research is to study the effect that the values of the thresholds have on the quality of the solution and to identify features of a problem instance that determine that effect.

Heuristics

Vertex selection and color selection are the two key components of our simple, constructive algorithm, and our strategies for both are flexible in the varied ways they use new heuristics and variations of the traditional ones. Our current implementation uses 10 'primitive' heuristics for selecting the next vertex to be coloured and four to select a colour for that vertex.

Ten Primitive Vertex-Selection Heuristics

Our colouring strategies are based on the classical and intuitive idea that the most troublesome vertices should be coloured first. Some of the commonly used heuristics based on this idea have been largest saturated degree, largest degree, and largest weighted degree.

We use variations of these, and we introduce some new ones that focus more on the number of bad edges and the number of bad colours. Some of these new heuristics rely on the information kept in each vertex's colour-penalties vector, while others use information tied to the edges incident on each vertex. The primitive heuristics on which our vertex selectors are based are:

0. *Maximum number of bad-conflict edges to uncoloured neighbors* – vertices having the most bad-conflict edges among their incident edges to uncoloured neighbors.
1. *Maximum number of bad-conflict colours* – vertices having the most bad-conflict colours. For the Toronto data set, this heuristic reduces to largest saturation degree.
2. *Maximum number of bad-proximity colours* – vertices having the most bad-proximity colours.
3. *Maximum conflict sum* – vertices with the largest sum of their conflict colour penalties.
4. *Maximum proximity sum* – vertices with the largest sum of their proximity colour penalties.
5. *Maximum conflict degree to uncoloured neighbors* – vertices whose incident edges to uncoloured neighbors have the largest sum of the conflict severities.
6. *Maximum number of bad-conflict edges* – vertices having the most bad-conflict edges among their incident edges. For the Toronto data set, this reduces to largest degree (since *every* edge is considered a bad-conflict edge).
7. *Maximum number of bad-intersect edges to uncoloured neighbors* – vertices having the most bad-intersect edges among their incident edges to uncoloured neighbors.
8. *Maximum intersect degree to uncoloured neighbors* – vertices whose incident edges to uncoloured neighbors have the largest sum of the intersection sizes.
9. *Maximum number of bad colours* – a consolidation of heuristics 1 and 2; a bad colour is one whose conflict penalty or whose proximity penalty exceeds its respective threshold.

Observe that heuristic 7 may be better at evaluating the difficulty of a vertex than its sum counterpart, heuristic 8. To illustrate, suppose that the edge weights in Figure 4 represent intersection size and that all neighbors of vertices v1 and v2 are uncoloured. Then heuristic 8 would select v1, whereas, for any bad-intersect-edge threshold greater than one, heuristic 7 would select v2, which appears to be more difficult. A similar observation can be made for heuristic 2 versus heuristic 4.

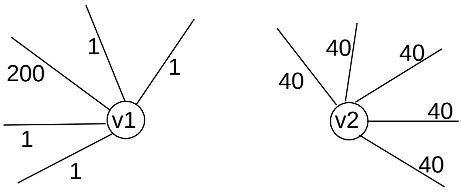


Figure 4: Heuristic 8 would select v1 before v2.

Four Primitive Colour-Selection Heuristics

Given a vertex v that has been selected, the primitive heuristics that we use to choose a colour for v are:

0. *Minimum conflict penalty* – a colour that has minimum conflict penalty for vertex v.
1. *Minimum proximity penalty* – a colour that has minimum proximity penalty for vertex v.
2. *Least bad for neighbors with respect to conflict penalty* – a colour which when assigned to v causes the fewest good-to-bad conflict penalty switches for the uncoloured neighbors of v.
3. *Least bad for neighbors with respect to proximity penalty* – same as heuristic 2 but with respect to proximity penalty.

Combining Heuristics

One of the innovations of our model and implementation is the ability to combine any number of the primitive heuristics to form compound vertex selectors and compound colour selectors. A compound vertex selector starts with one of the 10 primitive vertex-selection heuristics listed above. Typically there will be several vertices identified as the most difficult with respect to that heuristic. This subset of vertices is then narrowed down by applying a second primitive heuristic, and so on. Thus, a compound vertex selector consists of a sequence of primitive heuristics, where all but the first one in the sequence, is regarded as a tiebreaker for the ones before it. Once the subset of vertices is pared down by the combination of heuristics, some vertex is chosen from the subset (typically the first one in the list). Compound colour selectors are similarly constructed from the four primitive colour-selection heuristics listed above.

Switching Selectors in the Middle of a Coloring

Another feature of our model is the ability to switch from one combination of heuristics to another at various stages of the colouring. Including this feature was motivated by the general observation that the effectiveness of a heuristic is likely to change as the colouring progresses. The primitive vertex-selection heuristic 1 is perhaps the simplest illustration of this behavior. As we mentioned earlier, this heuristic is essentially the traditional saturation degree, which has proven to be among the most preferred heuristics for classical graph colouring. However, applying heuristic 1 in the very early stages of a colouring will produce a huge number of ties. Moreover, early in a colouring, the only vertices with any bad-conflict colours will tend to be those few that have neighbors that have already been coloured. Thus, until several vertices are coloured, the order in which they are selected will tend toward a simple breadth-first order and not be an effective predictor of the difficult-to-colour vertices.

Accordingly, the compound vertex selectors used early in the colouring process begin with a primitive heuristic based on the weights of incident edges (e.g., heuristic *0*). Then, after a designated number of vertices have been selected and colored, we switch to a compound selector that begins with heuristic 1 when it is more likely to be a stronger predictor of the difficulty of a vertex.

Vertex Partitioning

A final innovation involves a preprocessing step that partitions the vertex set and allows us to reduce the amount of computation without incurring additional conflict penalties. The preprocessing is based on the following simple observation. If *v* is a vertex with degree less than *k*, and *v* initially has *k* colours available, then *v* can safely be left until last to colour, since it will always have at least one non-conflict colour available, independent of how its neighbors are coloured and how heavy the edge-weights are between *v* and its neighbors.

The preprocessing uses an iterative partitioning algorithm that places all vertices whose colouring can be done last into the easiest-to-colour subset, say S_1. Next, for each vertex in S_1, we calculate a reduced (quasi-) degree of each of its neighbors and put all vertices whose reduced degree is less than the number of colours available into the next-easiest-to-colour subset, S_2. Again, as long as a vertex in S_2 is coloured before any of its neighbors in S_1, it can safely be left uncoloured until its other neighbors are coloured. The process continues until no additional vertices can be removed from the ‘hardest’ subset and the vertices in that last subset of the partition must be coloured first using the specified selection criteria.

As long as the subsets are done in order (last to first), vertices in all subsets except for the hardest one can be selected arbitrarily with no possibility of incurring a conflict penalty. One simply chooses an available colour, whose existence is guaranteed by the construction. Thus, in a fairly sparse graph, computation can be considerably reduced. Notice that because any penalties that result from the colouring occur in the process of colouring the hardest cell, any local improvement algorithms could be applied only to that set of vertices before moving on to colour the rest of the graph, again without incurring additional penalties at a later stage.

Another potential advantage to this partitioning strategy is that the vertex-selection process after the hardest subset has been coloured can be based solely on proximity considerations.

Some Preliminary Results

We present the preliminary results of applying our approach on the Toronto benchmarks, which is available at ftp://ftp.mie.utoronto.ca/pub/carter/testprob/. This dataset was first introduced in (Carter, Laporte, and Lee 1996), and since then has been extensively studied using a wide range of algorithms in the literature. We set the number of colors equal to the number of time slots in the Toronto dataset. Due to the fact that two versions of the datasets have been circulated under the same name in the last ten years, we have renamed the problems in (Qu et al. 1996). We used version I of the data in our experiments.

Testing is ongoing and much more needs to be done. However, we can make some initial observations.

Table 1 presents the best results we have obtained so far.

Although we haven't fully tested it yet, partitioning appears to improve solution quality most of the time. Except for the "sta83 I" problem instance, all results in column 2 of the table were produced using the partitioning pre-processing.

We obtained them using the following two groups of three compound vertex selectors:

vs1: 0 7 8 1 2 4 | 1 0 2 4 7 8 | 2 4 7 8
vs2: 0 7 8 9 4 | 9 0 7 8 2 4 | 2 4 7 8

The numbers refer to the primitive vertex-selection heuristics introduced earlier, and the vertical lines separate the three compound selectors that form each group. The first compound selector in a group is applied to the hardest subset until a designated fraction (the *switch fraction*) of the vertices have been selected and coloured. Then the second compound selector is applied to the rest of the hardest subset. Finally, the third selector, which consists of the four proximity-related primitive heuristics, is applied to the remaining (non-hard) vertices.

We used the following two groups of two compound color selectors:

cs0: 0 1 2 3 | 0 1 3
cs1: 0 2 3 1 | 0 3 1

The first compound selector in each group was applied to the entire subset of hardest-to-color vertices, and the second one was applied to the rest of the vertices.

As we described earlier, the thresholds for a bad-proximity color and a bad-intersect edge were set equal to the average intersection size times two different constant multipliers. In the table, PC is the multiplier for the bad-proximity color, and IE is the one for the bad-intersect edge.

The Settings column gives the values of the switch fraction and the multipliers, PC and IE, and indicates the vertex and color selectors used to produce the given result.

Problem	**Best results**	**Settings switch | PC | IE | vs | cs**	**Best reported**
car91 I	5.22	1/23 | 90 | 1 | vs2 | cs0	4.97
car92 I	4.40	1/13 | 126 | 2 | vs2 | cs0	4.32
ear83 I	39.28	1/5.2 | 115.5 | 1,2 | vs2 | cs0	36.16
hec92 I	12.35	1/5 | 16 | 1,2 | vs1 | cs0	10.8
kfu93 I	19.04	1/14 | 134 | 1,2 | vs2 | cs0	14.0
lse91	12.05	1/32 | 192 | 1,2 | vs2 | cs0	10.5
rye92	10.21	1/28 | 133.5 | 2 | vs2 | cs0	7.3
sta83 I	163.05	1/26.5 | 81 | 1 | vs2 | cs1	158.19
tre92	8.62	1/39 | 207 | 20 | vs2 | cs0	8.38
ute92	3.62	1/16 | 50 | 1,2 | vs1 | cs0	3.36
uta92 I	30.60	1/5 | 369 | 1,2 | vs2 | cs1	25.8
yor83 I	42.05	1/17 | 340 | 2 | vs2 | cs0	39.8

Table 1. Best results with the corresponding settings for Toronto benchmarks.

Results from Table 1 demonstrate that for vertex selection, vs2 outperforms vs1; 10 of the 12 best results were achieved using vs2. Changing threshold values for badness and changing the switch point between the first and second compound vertex selector clearly affect the performance of our algorithm.

In Table 1, we also gave the best results reported in the literature which used different constructive methods. Although our totals for proximity penalty are, on the average, 13% worse than the best ones reported, we believe our approach still holds promise, particularly in view of the fact that it is, at the moment, a one-pass algorithm without any backtracking or local improvement. The best results reported in the last column were by different approaches cited in the literature. No single algorithm outperformed others on all problems tested here.

In general, these preliminary results indicate that the performance of the algorithm is sensitive to the settings of the switch points and thresholds. Although we have some initial observations on which settings perform better on which Toronto problems, the setting of these parameters in relation to particular problems is not clear. More research effort needs to be spent to develop more intelligent mechanisms to adaptively choose these settings for different problems.

One of our future directions is to use *heuristics* to choose how to construct the combinations of heuristics. This *hyper-heuristic* approach (see Burke et al. 2003) has been applied successfully in a range of scheduling and optimization problems, including timetabling. It is well known in meta-heuristics research that different heuristics perform better on different problems, or even different instances of the same problem. One of the research challenges is concerned with the *automatic* design of heuristics in solving a wider range of problems. Developing an automatic algorithm that can intelligently operate on a search space of vertex and colour selectors, switch point selectors and threshold settings will become one of our primary research efforts in the future.

Features of the Model Not Being Used Yet

There are some features of our model not used in our current implementation that add to its robustness.

Our model can handle *pre-colored* vertices, that is, exams that must be assigned to certain time slots. Furthermore, if certain time slots are forbidden for a particular exam (for example, the professor is only available on certain days and times), then this can easily be handled by setting an initial nonzero penalty for the relevant color.

As we noted earlier, each color, which represents a time slot, can have attributes associated with fairly general information, like start time, duration and/or finish time. For this paper we used only a single attribute, an integer value between zero and the maximum number of time slots in use, since we were testing our implementation on the Toronto benchmark problems.

Ongoing and Future Work

The robust model presented in this paper can be easily extended or integrated with other techniques to develop more advanced and powerful algorithms. We give below some possible (and ongoing) research directions.

- Study the effects of varying the switch points, the badness threshold values, and the use of different heuristic combinations. In the context of *hyper-heuristics*, there are a number of different search spaces to consider:
  - The set of all the combinations of one or more of the primitive vertex selectors and of the color-selectors.
  - For a given group of compound vertex selectors, the set of all switch points.
  - For a given group of compound vertex selectors, the set of threshold values for badness.
- In the context of *case-based reasoning*, test heuristic combinations, thresholds, and switch points with randomly generated problem instances that are in the Toronto format to see if certain performance patterns emerge. Previous work on using case-based reasoning (see Burke, Petrovic and Qu, 2006) to intelligently select graph colouring heuristics demonstrated that there are significant, wide-ranging possibilities for research in knowledge-based heuristic design.
- Adding a backtracking component to the algorithm is likely to lower the total proximity penalty. For instance, when every colour assignment for a selected vertex incurs a proximity penalty above some threshold, the algorithm un-colours or re-colours some other vertex in order to reduce the selected vertex's proximity penalty.
- Write an improvement method that takes a given colouring produced by our algorithm and looks for vertices whose colours can be changed to decrease the total proximity penalty while maintaining feasibility.
- With the current implementation, we have not yet made full use of the varying conflict severity of edges, nor have we allowed any trade-off between conflict penalty and proximity penalty. In timetabling situations where conflicts must be tolerated, the end-user might specify that a certain amount of conflict penalty is equivalent to a certain amount of proximity penalty, e.g., a proximity violation involving 50 students equals a conflict involving one student. This might lead naturally to a single objective function to be minimized.
- As we mentioned at the start, the model can be adapted to a variety of scenarios, in which a number of parameters would be specified interactively by the end user through an appropriate interface. Follow-up work will include building such an interface.

Acknowledgements

The research for this paper was supported by Nottingham University, UK, the Engineering and Physics Science Research Council (EPSRC), UK, and an Ashforth Grant from Rollins College, USA.

References

Broder, S., Final Examination Scheduling, Comm. of the ACM 7 (1964), 494-498.

Brelaz, D., New methods to color the vertices of a graph. Comm. of the ACM 22 (1979), 251-256.

Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P. and Schulenburg, S.: Hyperheuristics: an Emerging Direction in Modern Search Technology. In: Glover, F. and Kochenberger, G.: Handbook of Metaheuristics, 457-474, 2003.

Burke, E. K., Kingston, J. H., and de Werra, D., Applications to Timetabling, In: J. L. Gross and J. Yellen (eds.) The Handbook of Graph Theory, Chapman Hall/CRC Press, (2004), 445-474.

Burke, E.K., McCollum, B., Meisels, A., Petrovic, S. and Qu, R.: A Graph-Based Hyper Heuristic for Timetabling Problems. European Journal of Operational Research, 176 (2007) 177-192.

Burke, E.K., Petrovic, S., and Qu R., Case Based Heuristic Selection for Timetabling Problems. Journal of Scheduling, 9 (2006) 115-132.

Carter, M. W., A Survey of Practical Applications of Examination Timetabling Algorithms, Operations Research 34 (1986), 193-201.

Carter, M. W., Laporte, G., and Lee, S., Examination Timetabling: Algorithmic Strategies and Applications, J. of the Operations Research Society 47 (1996), 373-383.

de Werra, D., An Introduction to Timetabling, Euro. J. Oper. Res. 19 (1985), 151-162.

Kiaer, L., and Yellen, J., Weighted Graphs and University Timetabling, Computers and Operations Research Vol. 19, No. 1 (1992a), 59-67.

Kiaer, L., and Yellen, J., Vertex Coloring for Weighted Graphs With Application to Timetabling, Technical Report Series – RHIT, MS TR 92-12 (1992b).

Krarup, J., and de Werra, D., Chromatic Optimisation: Limitations, Objectives, Uses, References, Euro. J. Oper. Res. 11 (1982), 1-19.

Mehta, N. K., The Application of a Graph Coloring Method to an Examination Scheduling Problem, Interfaces 11 (1981), 57-64.

Neufeld, G. A. and Tartar, J., Graph Coloring Conditions for the Existence of Solutions to the Timetable Problem, Comm. of the ACM 17 (1974), 450-453.

Papadimitriou, C. H. and Steiglitz, K., Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, 1982.

Qu, R., Burke, E.K., McCollum, B., Merlot, L. T. G., and Lee, S. Y., A survey of Search Methodologies and Automated Approaches for Examination Timetabling, Technical Report, NOTT-CS-TR-2006-4 (2006).

Schaerf, A., A Survey of Automated Timetabling, Artificial Intelligence Review 13 (1999), 87-127.

Schmidt, G., and Strohlein, T., Timetable Construction--an Annotated Bibliography, The Computer Journal 23 (1980), 307-316.

Welsh, D. J. A., and Powell, M. B., An Upper Bound for the Chromatic Number of a Graph and its Application to Timetabling Problems, The Computer Journal 10 (1967), 85-86.

Wood, D. C., A System for Computing University Examination Timetables, The Computer Journal 11 (1968), 41-47.

A Multi-Component Framework for Planning and Scheduling Integration

Amedeo Cesta, Simone Fratini and **Federico Pecora**
ISTC-CNR, National Research Council of Italy
Institute for Cognitive Science and Technology
Rome, Italy
{name.surname}@istc.cnr.it

Abstract

This paper presents our recent work on OMPS, a new timeline-based software architecture for planning and scheduling whose features support software development for space mission planning applications. The architecture is based on the notions of domain components and is deeply grounded on constraint-based reasoning. Components are entities whose properties may vary in time and which model one or more physical subsystems which are relevant to a given planning context. Decisions can be taken on components, and constraints among decisions modify the components' behaviors in time.

Introduction

This paper describes OMPS, the Open Multi-component Planning and Scheduling architecture. OMPS implements a timeline-driven solving strategy. The choice of using timelines lies in their suitability for real-world problem specifications, particularly those of the space mission planning context. Furthermore, timelines are very close to the operational approach adopted by human planners in current space mission planning. Previous timeline-based approaches have been described in (Muscettola *et al.* 1992; Muscettola 1994; Cesta & Oddi 1996; Jonsson *et al.* 2000; Frank & Jónsson 2003; Smith, Frank, & Jonsson 2000). We are evolving from our previous work on a planner called OMP (Fratini & Cesta 2005) in which we have proposed a uniform view of state variables and resources timelines to integrate Planning & Scheduling (P&S). While the OMP experience lead to a proof of concept solver for small scale demonstration, the current development of OMPS is taking place within the Advanced Planning and Scheduling Initiative (APSI) of the European Space Agency (ESA). This has lead to a a substantial effort both in re-engineering and in extending our previous work.

The general goal in OMPS is to provide a development environment for enabling the design and implementation of mission planning decision support systems to be used by ESA staff. OMPS also inherits our previous experience in developing planning and scheduling support tools for ESA, namely with the MEXAR, MEXAR2 and RAXEM systems (Cesta *et al.* 2007), currently in active duty at ESA's control center. Our aim within APSI is to generalize the approach to mission planning decision support by creating a software framework that facilitates product development.

The OMPS architecture is not only influenced by constraint-based reasoning work, but introduces also the notion of domain *components* as a primitive entity for knowledge modeling. Components are entities whose properties may vary in time and which model one or more physical subsystems which are relevant to a given planning context. Decisions can be taken on components, and constraints among decisions modify the components' behaviors in time. Components provide the means to achieve modular decision support tool development. A component can be designed to incorporate into a constraint-based reasoning framework entire decisional modules which have been developed independently. The underlying philosophy of OMPS is to provide a development environment within which different, independently developed reasoning modules can be integrated seamlessly. It is useful to see a component as an entity having both *static* and *dynamic* aspects. Static descriptions are used to describe "what a component is", e.g., the static property of a light bulb is that it can be "on" or "off". Dynamic properties are instead those features which define how the static properties of the component may vary over time, e.g., a light bulb can go from "on" to "off" and vice-versa.

It is tempting to associate components to the concept of state variable *a la* HSTS (Muscettola *et al.* 1992; Muscettola 1994). The reason for not doing so is that a state variable models an entity with static properties. The way this entity can change over time is typically specified through constraints on the possible transitions and durations of the states (e.g., through a timed automaton). A component as we define it here represents a more general concept: its behavior over time can be determined by non-trivial reasoning which is *internal* to the component itself. This distinction is important, as it provides a way to seamlessly incorporate into the OMPS reasoning framework objects which are themselves capable of modifying their behavior according to non-trivial processes, such as sophisticated reasoning algorithms.

This paper is organized as follows. First, we define the basic building block, namely the component, providing examples which show how such an entity can be instantiated to represent a "classical" state variable, a resource, or even a more complex object whose temporal behavior can be described according to its own "internal dynamics". Second, we describe the notion of decision on a component. Again, we provide examples to show how this concept is instantiated on different common types of components. Third, we introduce the concepts of timeline and domain theory, the former providing the driving feature of the solving approach, the latter describing how components interact, and how decisions taken on components affects other components. Finally, we briefly illustrate the solving strategy implemented in the current OMPS framework and provide an example. It is worth saying that this paper describes the general approach underlying the OMPS architecture. We do not dwell on the theoretical aspects underlying the architecture, for which the interested reader is referred to (Fratini 2006).

Components and Behaviors

An intrinsic property of *components* is that they evolve over time, and that decisions can be taken on components which alter their evolution. In OMPS, a component is an entity that has a set of possible temporal evolutions over an interval of time $\mathcal{H}$. The component's evolutions over time are named *behaviors*. Behaviors are modeled as temporal functions over $\mathcal{H}$, and can be defined over continuous time or as stepwise constant functions of time.

In general, a component can have many different behaviors. Each behavior describes a different way in which the component's properties vary in time during the temporal interval of interest. It is in general possible to provide different representations for these behaviors, depending on (1) the chosen temporal model (continuous vs. discrete, or time point based vs. interval based), (2) the nature of the function's range $\mathcal{D}$ (finite vs. infinite, continuous vs. discrete, symbolic vs. numeric) and (3) the type of function which describes a behavior (general, piecewise linear, piecewise constant, impulsive and so on).

Not every function over a given temporal interval can be taken as a valid behavior for a component. The evolution of components in time is subject to "physical" constraints (or approximations thereof). We call *consistent* behaviors the ones that actually correspond to a possible evolution in time according to the real-world characteristics of the entity we are modeling. A component's consistent behaviors are defined by means of *consistency features*. In essence, a consistency feature is a function f^C which determines which behaviors adhere to physical attributes of the real-world entity modeled by the component.

It is in general possible to have many different representations of a component's consistency features: either explicit (e.g., tables or allowed bounds) or implicit (e.g., constraints, assertions, and so on). For instance, let us model a light bulb component. A light bulb's behaviors can take three values: "on", "off" and "burned". Supposing the light bulb cannot be fixed, a rule could state that any behavior that takes the value "burned" at a time t is consistent if and only if such a value is taken also for any time $t' > t$. This is a declarative approach to describing the consistency feature f^C. Different actual representations for this function can be used, depending also on the representation of the behavior.

A few more concrete examples of components and their associated consistency features are the following.

State variable. *Behaviors*: piecewise constant functions over a finite, discrete set of symbols which represent the *values* that can be taken by the state variable. Each behavior represents a different sequence of values taken by the component. *Consistency Features*: a set of sequence constraints, i.e., a set of rules that specify which transitions between allowed values are legal, and a set of lower and upper bounds on the duration of each allowed value. The model can be for instance represented as a timed automaton (Alur & Dill 1994) (e.g., the three state variables in Figure 2).

Note that a distinguishing feature of a state variable is that not all the transitions between its values are allowed.

Resource (renewable). *Behaviors*: integer or real functions of time, piecewise, linear, exponential or even more complex, depending on the model you want to set up. Each behavior represents a different profile of resource consumption. *Consistency Feature*: minimum and maximum availability. Each behavior is consistent if it lies between the minimum and maximum availability during the entire planning interval.

Note that a distinguishing feature of a resource is that there are bounds of availability.

In general, the component-based approach allows to accommodate a pre-existing solving component into a larger planning problem. For instance, it is possible to incorporate the MEXAR2 application (Cesta *et al.* 2007) as a component, the consistency property of which is not computed directly on the values taken by the behaviors, but as a function of those behaviors[1].

Component Decisions

Now that we have defined the concept of component as the fundamental building block of the OMPS architecture, the next step is to define how component behaviors can be altered (within the physical constraints imposed by consistency features).

We define a *component decision* as a pair $\langle \tau, \nu \rangle$, where τ is a given *temporal element*, and ν is a *value*. Specifically, τ can be:

- A time instant (TI) t representing a moment in time.
- A time interval (TIN), a pair of TIs defining an interval $[t_s, t_e)$ such that $t_e > t_s$.

The specific form of the value ν depends on the type of component on which the decision is defined. For instance, this can be an amount of resource usage for a resource component, or a disjunction of allowed values for a state variable.

Regardless of the type of component, the value of any component decision can contain *parameters*. In OMPS, parameters can be numeric or enumerations, and can be used to express complex values, such as "transmit(?bitrate)" for a

[1] Basically, it is computed as the difference between external uploads and the downloaded amount stated by the values taken by the behaviors. See (Cesta *et al.* 2007) for details on the MEXAR2 application.

state variable which models a communications system. Further details on value parameters will be given in the following section.

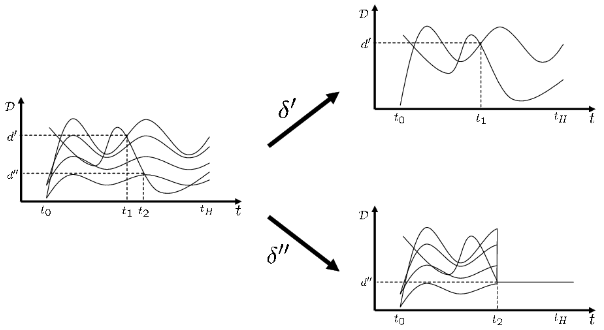


Figure 1: The update function computes the results of a decision on a component's set of behaviors. The figure exemplifies this effect given the two decisions: δ' imposes a value d' for the behaviors of the component in the time instant t_1; δ'' imposes that the values of all behaviors converge to d'' after time instant t_2.

Overall, a component decision is something that happens somewhere in time and modifies a component's behaviors as described by the value ν. In OMPS, the consequences of these decisions are computed by the components by means an *update function* f^U. This is a function which determines how the component's behaviors change as a consequence of a given decision. In other words, a decision changes a component's set of behaviors, and f^U describes how. A decision could state for instance "keep all the behaviors that are equal to d' in t_1" and another decision could state "all the behaviors must be equal to d'' after t_2". Given a decision on a component with a given set of behaviors, the update function computes the resulting set (see Figure 1).

In the following, we instantiate the concept of decision for the two types of components we have introduced so far.

State variable. *Temporal element*: a TIN. *Value*: a subset of values that can be taken by the state variable (the range of its behaviors) in the given time frame. *Update Function*: this kind of decision for a state variable implies the choice of values in a given time interval. In this case the subset of values are meant as a disjunction of allowed values in the given time interval. Applying a decision on a set of behaviors entails that all behaviors that do not take any of the chosen values in the given interval are excluded from the set.

Resource (renewable). *Temporal element*: a TIN. *Value*: quantity of resource allocated in the given interval — a decision is basically an *activity*, an amount of allocated resource in a time interval. *Update Function*: the resource profile is modified by taking into account this allocation. Outside the specified interval the profile is not affected.

Domain Theory

So far, we have defined components in isolation. When components are put together to model a real domain they cannot be considered as reciprocally decoupled, rather we need to take into account the fact that they influence each other's behavior.

In OMPS, it is possible to specify such inter-component relations in what we call a *domain theory*. Specifically, given a set of components, a domain theory is a function f^{DT} which defines how decisions taken on one component affect the behaviors of *other* components. Just as a consistency feature f^C describes which behaviors are allowed with respect to the features of a single component, the domain theory specifies which of the behaviors belonging to all modeled components are concurrently admissible.

In practice, a domain theory is a collection of *synchronizations*. A synchronization essentially represents a rule stating that a certain decision on a given component (called the *reference* component) can lead to the application of a new decision on another component (called *target* component). More specifically, a synchronization has the form $\langle C_i, V\rangle \longrightarrow \langle C_j, V', R\rangle$, where: C_i is the reference component; V is the value of a component decision on C_i which makes the synchronization applicable; C_j is the target component on which a new decision with value V' will be imposed; and R is a set of *relations* which bind the reference and target decisions.

In order to clarify how such inter-component relationships are modeled as a domain theory, let us give an example.

Example 1 *The planning problem consists in deciding data transmission commands from a satellite orbiting Mars to Earth within given visibility windows. The spacecraft's orbits for the entire mission are given, and are not subject to planning. The fundamental elements which constitute the system are: the satellite's Transmission System (TS), which can be either in "transmit mode" on a given ground station or idle; the satellite's Pointing System (PS); and the satellite's battery (BAT). In addition, an external, uncontrollable set of properties is also given, namely Ground Station Visibility (GSV) and Solar Flux (SF). Station visibility windows are intervals of time in which given ground stations are available for transmission, while the solar flux represents the amount of power generated by the solar panels given the spacecraft's orbit. since the orbits are given for the entire mission, the power provided by the solar flux is a given function of time* $\mathrm{sf}(t)$. *The satellite's battery accumulates power through the solar flux and is discharged every time the satellite is slewing or transmitting data. Finally, it is required that the spacecraft's battery is never discharged beyond a given minimum power level (in order to always maintain a minimum level of charge in case an emergency manoeuvre needs to be performed).*

Instantiation this example into the OMPS framework thus equates to defining five components:

PS, TS and GSV. The spacecraft's pointing and transmission systems, as well as station visibility are modeled with three state variables. The consistency features of these state variables (possible states, bounds on their duration, and allowed transitions) are depicted in Figure 2. The figure also shows the synchronizations involving the three components: one states that the value "locked(?st3)" on

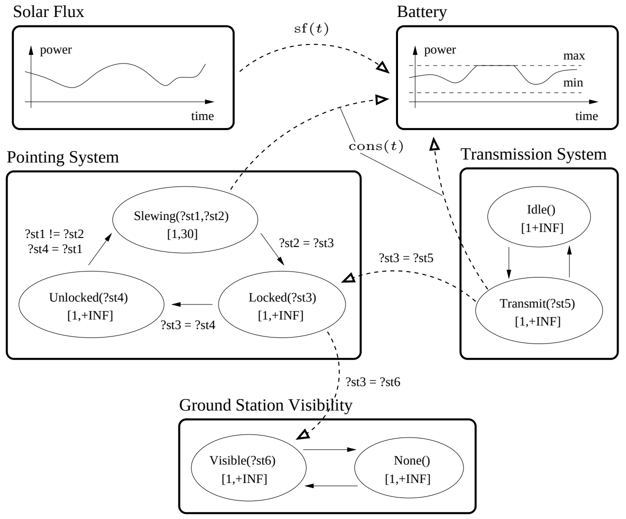


Figure 2: State variables and domain theory for the running example.

component PS requires the value "visible(?st6)" on component GSV (where ?st3 = ?st6, i.e., the two values must refer to the same station); another synchronization asserts that transmitting on a certain station requires the PS component to be locked on that station; lastly, both slewing and transmission entail the use of a constant amount of power from the battery.

SF. The solar flux is modeled as a reusable resource. Given that the flight dynamics of the spacecraft are given (i.e., the angle of incidence of the Sun's radiation with the solar panels is given), the profile of the solar flux resource is given function time $\mathrm{sf}(t)$ which is not subject to changes. Thus, decisions are never imposed on this component (i.e., the SF component has only one behavior), rather its behavior is solely responsible for determining power production on the battery (through the synchronization between the SF and BAT components).

BAT. The spacecraft's battery component is modeled as follows. Its consistency features are a maximum and minimum power level $(\max, \min)$, the former representing the battery's maximum capacity, the latter representing the battery's minimum depth of discharge. The BAT component's behavior is a temporal function $\mathrm{bat}(t)$ representing the battery's level of charge. Assuming that power consumption decisions resulting from the TS and PS components are described by the function $\mathrm{cons}(t)$, the update function calculates the consequences of power production $(\mathrm{sf}(t))$ and consumption on $\mathrm{bat}(t)$ as follows:

$$\mathrm{bat}(t) = \begin{cases} L_0 + \alpha \int_0^t (\mathrm{sf}(t) - \mathrm{cons}(t))dt \\ \quad \text{if } L_0 + \alpha \int_0^t (\mathrm{sf}(t) - \mathrm{cons}(t))dt \leq \max; \\ \max \\ \quad \text{otherwise.} \end{cases}$$

where L_0 is the initial charge of the battery at the beginning of the planning horizon and α is a constant parameter which approximates the charging profile.

In summary, we employ components of three types: state variables to model the PS, TS and GSV elements, a reusable resource to maintain the solar flux profile, and an ad-hoc component to model the spacecraft's battery. Notice that this latter component is essentially an extension of a reusable resource: whereas a reusable resource's update function is trivially the sum operator (imposing an activity on a reusable resource entails that the resource's availability is decreased by the value of the activity), the BAT's update function calculates the consequences of activities as per the above integration over the planning horizon.

Decision Network

The fundamental tool for defining dependencies among component decisions are *relations*, of which OMPS provides three types; *temporal*, *value* and *parameter* relations.

Given two component decisions, a *temporal relation* is a constraint among the temporal elements of the two decisions. A temporal relation among two decisions A and B can prescribe temporal requirements such as those modeled by Allen's interval algebra (Allen 1983), e.g., A EQUALS B, or A OVERLAPS [l,u] B.

A *value relation* between two component decisions is a constraint among the values of the two decisions. A value relation among two decisions A and B can prescribe requirements such as A EQUALS B, or A DIFFERENT B (meaning that the value of decision A must be equal to or different from the value of decision B). Notice that temporal relations can involve any two component decisions, e.g., an activity (a resource decision) should occur BEFORE a value choice (a state variable decision). Conversely, value relations are defined among decisions pertaining to components of the same type.

Lastly, a *parameter relation* among component decisions is a constraint among the values of the parameters of the two decisions. Such relations can prescribe linear inequalities between parameter variables. For instance, a parameter constraint between two decisions with values "available(?antenna, ?bandwidth)" and "transmit(?bitrate)" can be used to express the requirement that transmission should not use more than half the available bandwidth, i.e., ?bitrate $\leq 0.5\cdot$?bandwidth.

Component decisions and relations are maintained in a *decision network*: given a set of components $\mathcal{C}$, a decision network is a graph $\langle V, E\rangle$, where each vertex $\delta_C \in V$ is a component decisions defined on a component $C \in \mathcal{C}$, and each edge $(\delta_{C^i}^m, \delta_{C^j}^n)$ is a temporal, value or parameter relation among component decisions $\delta_{C^i}^m$ and $\delta_{C^j}^n$.

We now define the concepts of *initial condition* and *goal*.

An initial condition for our problem consists in a set of value choices for the GSV state variable. These decisions reflect the visibility windows given by the Earth's position with respect to the (given) orbit of the satellite. Notice that the allowed values of the GSV component are not references for a synchronization, thus they cannot lead to the insertion in the plan of new component decisions.

Conversely, a goal consists in a set of component decisions which are intended to trigger the solving strategy to exploit the domain theory's synchronizations to synthesize decisions. In our example, this set consists in value choices on the TS component which assert a desired number of "transmit(?st5)" values. Notice that these value choices can be allocated flexibly on the timeline.

In general, the characterizing feature of decisions which define an initial condition is that these decisions do not lead to application of the domain theory. Conversely, goals directly or indirectly entail the need to apply synchronizations in order to reach domain theory compliance. This mechanism is the core of the solving process described in the following section.

Reasoning About Timelines in OMPS

OMPS implements a solving strategy which is based on the notion of *timeline*. A timeline is defined for a component as an ordered sequence of its values. A component's timeline is defined by the set of decisions imposed on that component. Timelines represent the *consequences* of the component decisions over the time axis, i.e., a timeline for a component is the collection of all its behaviors as obtained by applying the f^U function given the component decisions taken on it.

The overall solving process implemented in OMPS is composed of three main steps, namely *domain theory application*, *timeline management* and *solution extraction*. More in detail, timeline management consists in *extraction*, *scheduling* and *completion*. Indeed, a fundamental principle of the OMPS approach is its *timeline-driven* solving process.

Domain Theory Application

Component decisions possess an attribute which changes during the solving process, namely whether or not a decision is *justified*. OMPS's domain application step consists in iteratively tagging decisions as justified according to the following rules (iterated over all decisions δ in the decision network):

1. If δ *unifies with another decision in the network*, then mark δ as justified;
2. If δ's value *unifies with the reference value of a synchronization in the domain theory*, then mark δ as justified and add the target decision(s) and relations to the decision network;
3. If δ *does not unify with any reference value in the domain theory*, mark δ as justified.

The previous definition of initial condition and goal can be understood in terms of domain theory application as follows: an initial condition is a set of component decisions whose justification follows trivially from the domain, i.e., it is the direct result of the application of step 3; a goal, on the other hand, is a set of component decisions whose justification leads to the application of synchronizations in the domain theory (i.e., step 2).

Timeline Management

Timeline management is a collection of procedures which are necessary to go from a set of decision network to a completely instantiated set of behaviors. These behaviors ultimately represent a solution to the planning problem. Timeline management may introduce new component decisions as well as new relations to the decision network. For this reason, the OMPS solving process iterates domain theory application and timeline management steps until the decision network is fully justified and a consistent set of behaviors can be extracted from all component timelines. The specific procedures which compose timeline management are *timeline extraction*, *timeline scheduling* and *timeline completion*. Before showing how these procedures are composed to form the core of our planning approach, we describe the three steps in detail.

Timeline Extraction. The outcome of the domain theory application step is a decision network where all decisions are justified. Nevertheless, since every component decision's temporal element (which can be a TI or TIN) is maintained in an underlying flexible temporal network, these decisions are not fixed in time, rather they are free to move between the temporal bounds obtained as a consequence of the temporal relations imposed on the temporal elements. For this reason, a timeline must be *extracted* from the decision network, i.e., the flexible placement of temporal elements implies the need of synthesizing a total ordering among floating decisions. Specifically, this process depends on the component for which extraction is performed. For a resource, for instance, the timeline is computed by ordering the allocated activities and summing the requirements of those that overlap. For a state variable, the effects of temporally overlapping decision are computed by intersecting the required values, to obtain (if possible) in each time interval a value which complies with all the decisions that overlap during the time interval.

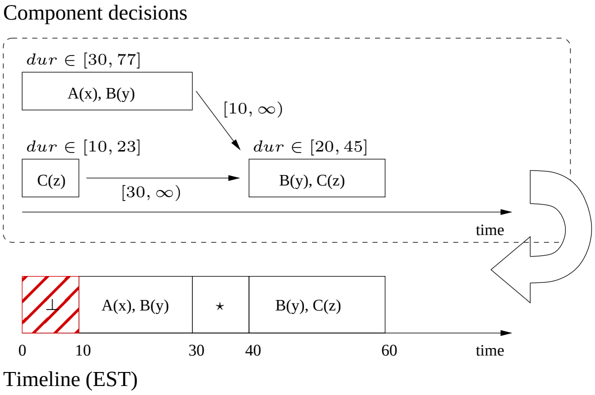


Figure 3: Three value choices on a state variable, and the resulting earliest start time (EST) timeline.

In the current implementation, we follow for every type of component an earliest start-time (EST) approach, i.e., we have a timeline where all component decisions are assumed to occur at their earliest start time and last the shortest time possible. Figure 3 shows the timeline extraction mechanism for a state variable. The example illustrates two properties of timelines, namely *flaws* and *inconsistencies*.

The first of these features depends on the fact that decisions imposed on the state variable do not result in a complete coverage of the planning horizon with decisions. This timeline in the figure contains what we call a *flaw* in the interval $[30, 40]$. A flaw is a segment of time in which no decision has been taken, thus the state variable within this segment of time is not constrained to take on certain values, rather it can, in principle, assume any one of its allowed values. The process of deciding which value(s) are admissible with respect to the state variable's internal consistency features (i.e., the component's f^C function) is clearly a non-trivial process. Indeed, this is precisely the objective of *timeline completion*.

In addition to flaws, *inconsistencies* can arise in the timeline. The nature of inconsistencies depends on the specific component we are dealing with. In the case of state variables, an inconsistency occurs when two or more value choices whose intersection is empty overlap in time. In the example above, this occurs in the interval $[0, 10]$. As opposed to flaws, inconsistencies do not require the generation of additional component decisions, rather they can be resolved by posting further temporal constraints. For instance, the above inconsistency can be resolved by imposing a BEFORE constraint which forces (C(z)) to occur after (A(x), B(y)). In the case of the BAT component mentioned earlier, an inconsistency occurs when slewing and/or transmission decisions have lead to a situation in which $\mathrm{bat}(t) \leq \min$ for some $t \in \mathcal{H}$. As in the previous example, BAT inconsistencies can be resolved by posting temporal constraints between the over-consuming activities. In general, we call the process of resolving inconsistencies *timeline scheduling*.

Timeline Scheduling. The scheduling process deals with the problem of resolving inconsistencies. Once again, the process depends on the component. For a resource, activity overlapping results in an inconsistency if the combined usage of the overlapping activities requires more than the resource's capacity. For a state variable, any overlapping of decision that requires a conflicting set of decisions must be avoided. The timeline scheduling process adds constraints to the decision network to avoid such inconsistencies through a constraint posting algorithm (Cesta, Oddi, & Smith 2002).

Timeline Completion. This process is required for components such as state variables, where it is required that any interval of time in a solution is covered by a decision (this is trivially true for reusable resources as we have defined them in this paper). If it is not possible to force an ordering among decisions in such a way that entire planning horizon is decided, then a flaw completion routine is triggered. This step adds new decisions to the plan.

Solution Extraction

Once domain application and timeline management have successfully converged on a set of timelines with no inconsistencies or flaws, the next step is to extract from the timelines one or more consistent behaviors. Recall that a behavior is one particular choice of values for each temporal segment in a component's timeline. The previous domain theory application and timeline management steps have filtered out all behaviors that are not, respectively, consistent with respect to the domain theory and the components' consistency features. Behavior extraction deals with the problem of determining a consistent set of fully instantiated behaviors for every component. Since every segment of a timeline potentially represents a disjunction of values, behavior extraction must choose specific behaviors consistently. Furthermore, not all values in timeline segments are fully instantiated with respect to parameters, thus behavior extraction must also take into account the consistent instantiation of values across all components.

Overall Solving Process

In the current OMPS solver the previously illustrated steps are interleaved as sketched in Figure 4.

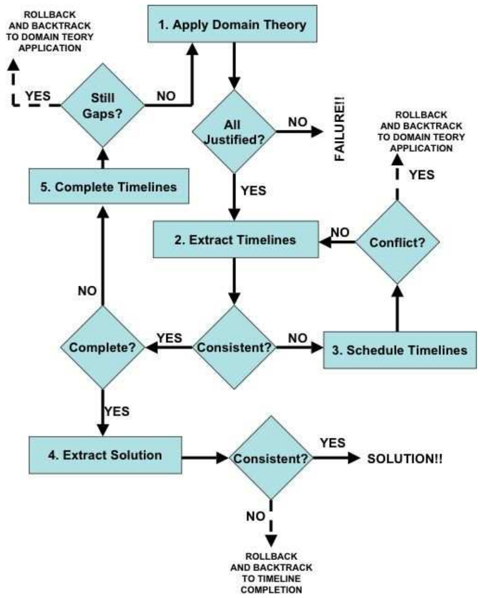


Figure 4: The OMPS solving process.

The first step in the planning process is domain theory application, whose aim is to support non-justified decisions. If there is no way to support all the decisions in the plan, the algorithm fails.

Once every decision has been supported, the solver tries to extract a timeline for each component. At this point, it can happen that some timelines are not consistent, meaning that there exists a time interval over which conflicting decisions overlap (an inconsistency). In such a situation, a scheduling step is triggered. If the scheduler cannot solve all conflicts, the solver backtracks directly to domain theory application, and searches for a different way of supporting goals.

If the solver manages to extract a conflict-free set of timelines, it then triggers a timeline-completion step on any timeline which is found to have flaws. It may happen that some timelines cannot be completed. In this case, the solver backtracks again to the previous domain theory application step, and again searches for a way of justifying all decisions. If the completion step succeeds for all timelines, the solver returns to domain theory application, as timeline completion has added decisions which are not justified.

Once all timelines are conflict-free and complete, the solver is ready to extract behaviors. If behavior extraction fails, the solver attempts to backtrack to timeline completion. This is because our currently implemented completion algorithm attempts to complete all incomplete timelines separately: thus it may easily happen that a completion over a timeline compromises behavior extraction on a different timeline (since values are linked with synchronizations). If this fails, the solver must return to domain theory application in order to search for a different plan altogether.

Finally, the whole process ends when the solver succeeds in extracting at least one behavior for each timeline. This collection of mutually consistent behaviors represents a fully instantiated solution to the planning problem.

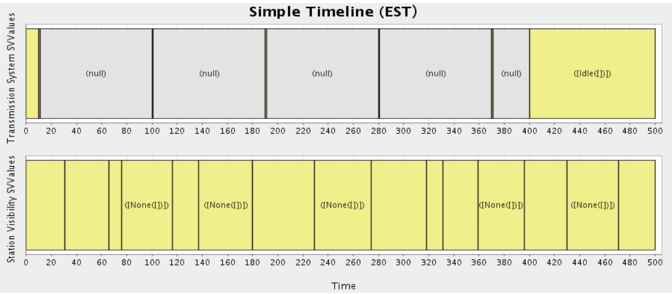


Figure 5: EST timelines for the TS and GSV state variables.

Going back to our running example, the timelines of the GSV and TS components resulting from the application of a set of initial condition and goal decisions are shown in Figure 5 (no initial decision or goal is specified for the PS component). Notice that the GSV timeline is fully defined, reflecting the fact that the GSV component is not controllable, rather it represents the evolution in time of station visibility given the fully defined flight dynamics of the satellite. The TS timeline contains five "transmit" value choices, through which we represent our goal. These value choices are allocated within flexible time bounds (the figure shows an EST timeline for the component, in which these decisions are anchored to their earliest start time and duration). As opposed to the GSV timeline, the TS timeline contains flaws, and it is precisely these flaws that will be "filled" by the solving algorithm. In addition, the application during the solving process of the synchronization between the GSV and PS components that will determine the construction of the PS's timeline (which is completely void of component decisions in the initial situation), reflecting the fact that it is necessary to point the satellite towards the visible target before initiating transmission.
The behaviors extracted from the TS and PS components' timelines after applying this solving procedure on our example are shown in Figure 6.

Related Work

The synthesis of OMPS is aimed at creating an extensible problem solving architecture to support development of different applications. It is worth making a comparison with other systems that, for different reasons, share the same goal with OMPS.

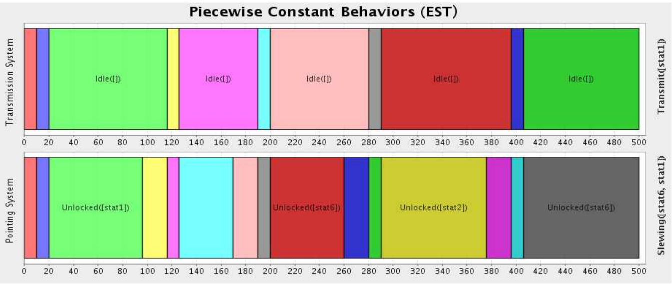


Figure 6: EST behaviors for the TS and PS state variables.

Similarly to OMPS's timelines, IxTeT (Ghallab & Laruelle 1994) follows a domain representation ontology based on state attributes which assume values in a given domain. Unlike OMPS in IxTeT system dynamics are represented with a STRIPS-like logical formalism. Resource reasoning is used as a conflict analyzer on top of the plan representation.

Visopt ShopFloor (Bartak 2002) is grounded on the the idea of working with dynamic scheduling problems where it is not possible to describe in advance activity sets that have to be scheduled. That is the same principle behind the integration of planning into scheduling done in both OMP and OMPS: to put a domain theory behind a scheduling problem to gain flexibility in managing tasks and goal driven problem solving. Dynamic aspects of the problem are described using resources with complex behaviors. These resources are close to our state variable, but they are managed using global constraints instead of a precedence constraint posting approach as we are currently doing. Moreover, although we are working on P&S integration we maintain a clear distinction between planning and scheduling at the level of modeling problem features.

HSTS (Muscettola *et al.* 1992; Muscettola 1994), has been the first to propose a modeling language with explicit representation of timelines, using the concept of state variables. In fact we are extending an HSTS-like state variables modeling language with a generic timeline oriented approach: in OMPS timelines represents not only state variable evolutions, but also multi-capacity and consumable resources, and may arrive to include generic components having temporal functions as behaviors. A clear difference w.r.t. HSTS is that in our approach we see different types of timelines as separate modules, while HSTS, and its derivatives RAX-PS and EUROPA, view resources as specialized state variables. Their view is certainly appealing but leaves the problem of integrating in a clean way multi-capacity resources open. In fact, while it is immediate to represent binary resources as state variables, it is quite difficult to model and handle cumulative resources. We believe that in these cases the best way is to exploit state of the art scheduling technologies hence our direction of seeing resources as an independent type of components.

Conclusions

In this article we have given a preliminary overview of OMPS a P&S system which follows a component-based, timeline-driven approach to planning and scheduling integration. The approach draws from and attempts to generalize our previous experience in mission planning tool development for ESA (Cesta *et al.* 2007) and to extend our previous work on the OMP planning system (Fratini & Cesta 2005).

A distinctive feature of the OMPS architecture is that it provides a framework for reasoning about any entity which can be modeled as a component, i.e., as a set of properties that vary in time. This includes "classical" concepts such as state variables (as defined in HSTS (Muscettola *et al.* 1992; Muscettola 1994) and studied also in subsequent work (Cesta & Oddi 1996; Jonsson *et al.* 2000; Frank & Jónsson 2003)), and renewable/consumable resources (Laborie 2003; Cesta, Oddi, & Smith 2002).

Another feature of the component-based architecture is the possibility to modularize the reasoning algorithms that are specific to each type of component within the component itself, e.g., profile-based scheduling routines for resource inconsistency resolution are implemented within the resource component itself. The more important consequence of this is the possibility to include previously implemented/deployed ad-hoc components within the framework. We have given an example of this in this paper with the battery component, which essentially extends a reusable resource. The ability to encapsulate potentially complex modules within OMPS components provides a strong added value in developing real-world planning systems. Specifically, this capability can be leveraged to include entire decisional modules which are already present in the overall decision process within which OMPS is deployed. An example is the MEXAR2 system (Cesta *et al.* 2007)[2], whose ability to solve the Mars Express memory dumping problem can be encapsulated into an ad-hoc component.

The ability to employ previously developed subsystems like MEXAR2 benefits decision support system development in a number of ways. From the engineering point of view, it facilitates the task of fast prototyping, providing a means to incorporate complex functionality by employing previously developed decision support aids. Also, this feature contributes to increasing the reliability of development prototypes, as existing components (especially in the context of ESA mission planning) have typically undergone intensive testing before being deployed. Second, the component-based architecture allows to leverage the efficiency of problem de-composition. Again, MEXAR2 provides a meaningful example, as it is a highly optimized decision support system for solving the very specific problem of memory dumping. Lastly, the ability to re-use components brings with it the advantage of preserving potentially crucial user interface paradigms, the re-engineering of which may be a strong deterrent for adopting innovative problem solving strategies.

[2] The MEXAR2 system is a specific decision support aid developed by the Planning and Scheduling Team which is currently in daily use within ESA's Mars Express mission.

Acknowledgments

The Authors are currently supported by European Space Agency (ESA) within the Advanced Planning and Scheduling Initiative (APSI). APSI partners are VEGA GmbH, ONERA, University of Milan and ISTC-CNR. Thanks to Angelo Oddi and Gabriella Cortellessa for their constant support, and to Carlo Matteo Scalzo for contributing an implementation of the battery component.

References

Allen, J. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26(11):832–843.

Alur, R., and Dill, D. L. 1994. A theory of timed automata. *Theor. Comput. Sci.* 126(2):183–235.

Bartak, R. 2002. Visopt ShopFloor: On the edge of planning and scheduling. In van Hentenryck, P., ed., *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP2002)*, LNCS 2470, 587–602. Springer Verlag.

Cesta, A., and Oddi, A. 1996. DDL.1: A Formal Description of a Constraint Representation Language for Physical Domains. In M.Ghallab, M., and Milani, A., eds., *New Directions in AI Planning*. IOS Press.

Cesta, A.; Cortellessa, G.; Fratini, S.; Oddi, A.; and Policella, N. 2007. An innovative product for space mission planning – an *a posteriori* evaluation. In *Proceedings of the 17th International Conference on Automated Planning & Scheduling (ICAPS-07).*

Cesta, A.; Oddi, A.; and Smith, S. F. 2002. A Constraint-based method for Project Scheduling with Time Windows. *Journal of Heuristics* 8(1):109–136.

Frank, J., and Jónsson, A. 2003. Constraint-based attribute and interval planning. *Constraints* 8(4):339–364.

Fratini, S., and Cesta, A. 2005. The Integration of Planning into Scheduling with OMP. In *Proceedings of the 2nd Workshop on Integrating Planning into Scheduling (WIPIS) at AAAI-05, Pittsburgh, USA*.

Fratini, S. 2006. *Integrating Planning and Scheduling in a Component-Based Perspective: from Theory to Practice*. Ph.D. Dissertation, University of Rome "La Sapienza", Faculty of Engineering, Department of Computer and System Science.

Ghallab, M., and Laruelle, H. 1994. Representation and Control in IxTeT, a Temporal Planner. In *Proceedings of the Second International Conference on Artifial Intelligence Planning Scheduling Systems*. AAAI Press.

Jonsson, A.; Morris, P.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in Interplanetary Space: Theory and Practice. In *Proceedings of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling (AIPS-00).*

Laborie, P. 2003. Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and new Results. *Artificial Intelligence* 143:151–188.

Muscettola, N.; Smith, S.; Cesta, A.; and D'Aloisi, D. 1992. Coordinating Space Telescope Operations in an Integrated Planning and Scheduling Architecture. *IEEE Control Systems* 12(1):28–37.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed., *Intelligent Scheduling*. Morgan Kauffmann.

Smith, D.; Frank, J.; and Jonsson, A. 2000. Bridging the gap between planning and scheduling. *Knowledge Engineering Review* 15(1):47–83.

Scheduling Monotone Interval Orders on Typed Task Systems

Benoît Dupont de Dinechin
STMicroelectronics STS/CEC
12, rue Jules Horowitz - BP 217. F-38019 Grenoble
benoit.dupont-de-dinechin@st.com

Abstract

We present a modification of the Leung-Palem-Pnueli parallel processors scheduling algorithm and prove its optimality for scheduling monotone interval orders with release dates and deadlines on Unit Execution Time (UET) typed task systems in polynomial time. This problem is motivated by the relaxation of Resource-Constrained Project Scheduling Problems (RCPSP) with precedence delays and UET operations.

Introduction

Scheduling problems on *typed task systems* (Jaffe 1980) generalize the parallel processors scheduling problems by introducing k types $\{\tau_r\}_{1\leq r\leq k}$ and $\sum_{1\leq r\leq k} m_r$ processors with m_r processors of type τ_r. Each operation O_i has a type $\tau_i \in \{\tau_r\}_{1\leq r\leq k}$ and may only execute on processors of type τ_i. We denote typed task systems with $\Sigma^k P$ in the α-field of the $\alpha|\beta|\gamma$ scheduling problem denotation (Brucker 2004).

Scheduling typed task systems is motivated by two main applications: resource-constrained scheduling in high-level synthesis of digital circuits (Chaudhuri, Walker, & Mitchell 1994), and instruction scheduling in compilers for VLIW processors (Dupont de Dinechin 2004). In high-level synthesis, execution resources correspond to the synthesized functional units, which are partitioned by classes such as adder or multiplier with a particular bit-width. Operations are typed by these classes and may have non-unit execution time. In compiler VLIW instruction scheduling, operations usually have unit execution time (UET), however on most VLIW processors an operation requires several resources for execution, like in the Resource-Constrained Project Scheduling Problems (RCPSP) (Brucker *et al.* 1999). In both cases, the pipelined implementation of functional units yield scheduling problems with precedence delays, that is, the time required to produce a value is larger than the minimum delay between two activations of a functional unit.

We are aware of the following work in the area of typed task systems. Jaffe (Jaffe 1980) introduces them to formalize instruction scheduling problems that arise in high-performance computers and data-flow machines, and studies the performance bounds of list scheduling. Jansen (Jansen 1994) gives a polynomial time algorithm for problem $\Sigma^k P|intOrder; p_i = 1|C_{max}$, that is, scheduling interval-ordered typed UET operations. Verriet (Verriet 1998) solves problem $\Sigma^k P|intOrder; c_i^j = 1; p_i = 1|C_{max}$ in polynomial time, that is, interval-ordered typed UET operations subject to unit communication delays.

Interval orders are a class of precedence graphs where UET scheduling on parallel processors is polynomial-time, while non-UET scheduling on 2 processors is strongly NP-hard (Papadimitriou & Yannakakis 1979). In particular, Papadimitriou and Yannakakis solve $P|intOrder; p_i = 1|C_{max}$ in polynomial-time. Scheduling interval orders with communication delays on parallel processors is also polynomial-time, as the algorithm by Ali and El-Rewini (Ali & El-Rewini 1992) solves $P|intOrder; c_i^j = 1; p_i = 1|C_{max}$. Verriet (Verriet 1996) further proposes a deadline modification algorithm that solves $P|intOrder; c_i^j = 1; r_i; p_i = 1|L_{max}$ in polynomial-time.

Scheduling interval orders with precedence delays on parallel processors was first considered by Palem and Simons (Palem & Simons 1993), who introduced monotone interval orders and solve $P|intOrder(mono\ l_i^j); p_i = 1|L_{max}$ in polynomial-time. This result is generalized by Leung-Palem-Pnueli algorithm (Leung, Palem, & Pnueli 2001).

In the present work, we modify the algorithm of Leung, Palem and Pnueli (Leung, Palem, & Pnueli 2001) in order to solve $\Sigma^k P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|-$ feasibility problems in polynomial time. The resulting algorithm thus operates on typed tasks, allows precedence delays, and handles release dates and deadlines. Thanks to these properties, it provides useful relaxations of the RCPSP with UET operations and precedence delays.

The Leung-Palem-Pnueli algorithm (Leung, Palem, & Pnueli 2001) is a parallel processors scheduling algorithm based on deadline modification and the use of lower modified deadline first priority in a Graham list scheduling algorithm. The Leung-Palem-Pnueli algorithm (LPPA) solves the following feasibility problems in polynomial time:

- $1|prec(l_i^j \in \{0,1\}); r_i; d_i; p_i = 1|-$
- $P2|prec(l_i^j \in \{-1,0\}); r_i; d_i; p_i = 1|-$
- $P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|-$
- $P|inTree(l_i^j = l); d_i; p_i = 1|-$

Here, the l_i^j are precedence delays with $p_i + l_i^j \geq 0$.

Presentation is as follows. In the first section, we extend the $\alpha|\beta|\gamma$ scheduling problem denotation and we discuss the Graham list scheduling algorithm (GLSA) for typed task systems. In the second section, we present our modified Leung-Palem-Pnueli algorithm (LPPA) and prove its optimality for scheduling monotone interval orders with release dates and deadlines on UET typed task systems in polynomial time. In the third section, we discuss the application of this algorithm to VLIW instruction scheduling.

Deterministic Scheduling Background

Machine Scheduling Problem Denotation

In parallel processors scheduling problems, an operation set $\{O_i\}_{1 \leq i \leq n}$ is processed on m identical processors. Each operation O_i requires the exclusive use of one processor for p_i time units, starting at its *schedule date* σ_i. Scheduling problems may involve *release dates* r_i and *due dates* d_i. This constrains the schedule date σ_i of operation O_i as $\sigma_i \geq r_i$ and there is a penalty whenever $C_i > d_i$, with C_i the *completion date* of O_i defined as $C_i \overset{\text{def}}{=} \sigma_i + p_i$. For problems where $C_i \leq d_i$ is mandatory, the d_i are called *deadlines*.

A *precedence* $O_i \prec O_j$ between two operations constrains the schedule with $\sigma_i + p_i \leq \sigma_j$. In case of *precedence delay* l_i^j between O_i and O_j, the scheduling constraint becomes $\sigma_i + p_i + l_i^j \leq \sigma_j$. The *precedence graph* has one arc (O_i, O_j) for each precedence $O_i \prec O_j$. Given an operation O_i, we denote $\mathsf{succ}O_i$ the set of direct successors of O_i and $\mathsf{pred}O_i$ the set of direct predecessors of O_i in the precedence graph. The set $\mathsf{indep}O_i$ contains the operations that are not connected to O_i in the undirected precedence graph.

Given a scheduling problem over operation set $\{O_i\}_{1 \leq i \leq n}$ with release dates $\{r_i\}_{1 \leq i \leq n}$ and deadlines $\{d_i\}_{1 \leq i \leq n}$, the *precedence-consistent release dates* $\{r_i^+\}_{1 \leq i \leq n}$ are recursively defined as $r_i^+ \overset{\text{def}}{=} \max(r_i, \max_{O_j \in \mathsf{pred}O_i}(r_j^+ + p_j + l_j^i))$. Likewise, the *precedence-consistent deadlines* $\{d_i^+\}_{1 \leq i \leq n}$ are recursively defined as $d_i^+ \overset{\text{def}}{=} \min(d_i, \min_{O_j \in \mathsf{succ}O_i}(d_j^+ - p_j - l_i^j))$.

Machine scheduling problems are denoted by a triplet $\alpha|\beta|\gamma$ (Brucker 2004), where α describes the processing environment, β specifies the operation properties and γ defines the optimality criterion. Values of α, β, γ include:

α : 1 for a single processor, P for parallel processors, Pm for the given m parallel processors. We denote typed task systems with k types by $\Sigma^k P$.

β : r_i for release dates, d_i for deadlines (if $\gamma = -$) or due dates, $p_i = 1$ for Unit Execution Time (UET) operations.

γ : $-$ for the feasibility, C_{max} or L_{max} for the minimization of these objectives.

The *makespan* is $C_{max} \overset{\text{def}}{=} \max_i C_i$ and the *maximum lateness* is $L_{max} \overset{\text{def}}{=} \max_i L_i : L_i \overset{\text{def}}{=} C_i - d_i$. The meaning of the additional β fields is:

$prec(l_i^j)$ Precedence delays l_i^j, assuming $l_i^j \geq -p_i$.

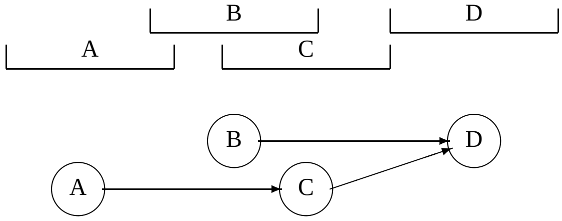


Figure 1: Set of intervals and the corresponding interval order graph.

$prec(l_i^j = l)$ All the precedence delays l_i^j equal l.

$inTree$ The precedence graph is an in-tree.

$intOrder(mono\, l_i^j)$ The precedence graph weighted by $w(O_i, O_j) \overset{\text{def}}{=} p_i + l_i^j$ is a monotone interval order.

An *interval order* is the transitive orientation of the complement of an interval graph (Papadimitriou & Yannakakis 1979) (see Figure 1). The important property of interval orders is that given any two operations O_i and O_j, either $\mathsf{pred}O_i \subseteq \mathsf{pred}O_j$ or $\mathsf{pred}O_j \subseteq \mathsf{pred}O_i$ (similarly for successors). This is easily understood by referring to the underlying intervals that define the interval order. Adding or removing operations without predecessors and successors to an interval order is still an interval order. Also, interval orders are transitively closed, that is, any transitive successor (predecessor) must be a direct successor (predecessor).

A *monotone interval order* graph (Palem & Simons 1993) is an interval order whose precedence graph (V, E) is weighted with a non-negative function w on the arcs such that, given any $(O_i, O_j), (O_i, O_k) \in E : \mathsf{pred}O_j \subseteq \mathsf{pred}O_k \Rightarrow w(O_i, O_j) \leq w(O_i, O_k)$. Monotone interval orders are motivated by the application of interval orders properties to scheduling problems with precedence delays.

Indeed, in scheduling problems with interval orders, the precedence arc weight considered between any two operations O_i and O_j is $w(O_i, O_j) \overset{\text{def}}{=} p_i$ with p_i the processing time of O_i. In case of monotone interval orders, the arc weights are $w(O_i, O_j) \overset{\text{def}}{=} p_i + l_i^j$ with l_i^j the precedence delay between O_i and O_j. An interval order graph where all arcs leaving any given node have the same weight is obviously monotone, so interval order precedences without precedence delays imply monotone interval order graphs.

Graham List Scheduling Algorithm Extension

The Graham list scheduling algorithm (GLSA) is a classic scheduling algorithm where the time steps are considered in non-decreasing order. For each time step, if a processor is idle, the highest priority operation available at this time is scheduled An operation is available if the current time step is not earlier than the release date and all direct predecessors have completed their execution early enough to satisfy the precedence delays. On typed task systems, the operation type must match the type of an idle processor.

The GLSA is optimal for $P|r_i; d_i; p_i = 1|-$ and $P|r_i; p_i = 1|L_{max}$ when using the earliest deadlines (or due dates) d_i first as priority (Brucker 2004) (Jackson's rule). This property directly extends to typed task systems:

Theorem 1 *The GLSA with Jackson's rule optimally solves* $\Sigma^k P|r_i; d_i; p_i = 1|-$ *and* $\Sigma^k P|r_i; p_i = 1|L_{max}$.

Proof: In typed task systems, operations are partitioned by processor type. In problem $\Sigma^k P|r_i; d_i; p_i = 1|-$ (respectively $\Sigma^k P|r_i; p_i = 1|L_{max}$), there are no precedences between operations. Therefore, optimal scheduling can be achieved by considering operations and processors of each type independently. For each type, the problem reduces to $P|r_i; d_i; p_i = 1|-$ (respectively $P|r_i; p_i = 1|L_{max}$), which is optimally solved with Jackson's rule. □

In this work, we allow precedences delays $l_i^j = -p_i \Rightarrow \sigma_i \leq \sigma_j$, that is, precedences with zero start-start time lags. Thus we extend the GLSA as follows: in cases of available operations with equal priorities, schedule first the earliest operations in the precedence topological sort order.

The Modified Leung-Palem-Pnueli Algorithm

Algorithm Description

The Leung-Palem-Pnueli algorithm (LPPA) is similar to classic UET scheduling algorithms on parallel processors like Garey & Johnson (Garey & Johnson 1976), in that it uses a lower modified deadlines first priority in a GLSA. Given a scheduling problem with deadlines $\{d_i\}_{1\leq i\leq n}$, *modified deadlines* $\{d'_i\}_{1\leq i\leq n}$ are such that $\forall i \in [1, n]$: $\sigma_i + p_i \leq d'_i \leq d_i$ for any schedule $\{\sigma_i\}_{1\leq i\leq n}$. The distinguishing feature of the LPPA is the computation of its modified deadlines, which we call *fixpoint modified deadlines*[1].

Precisely, the LPPA defines a *backward scheduling problem* denoted $B(O_i, S_i)$ for each operation O_i. An *optimal backward scheduling* procedure computes the latest possible schedule date σ'_i of operation O_i in each $B(O_i, S_i)$. Optimal backward scheduling of $B(O_i, S_i)$ is used to update the current modified deadline of O_i as $d'_i \leftarrow \sigma'_i + p_i$. This process of deadline modification is iterated over all problems $B(O_i, S_i)$ until a fixpoint of the modified deadlines $\{d^*_i\}_{1\leq i\leq n}$ is reached (Leung, Palem, & Pnueli 2001).

We modify the Leung-Palem-Pnueli algorithm (LPPA) to compute the fixpoint modified deadlines $\{d^*_i\}_{1\leq i\leq n}$ by executing the following procedure:

(i) Compute the precedence-consistent release dates $\{r^+_i\}_{1\leq i\leq n}$, the precedence-consistent deadlines $\{d^+_i\}_{1\leq i\leq n}$ and initialize the modified deadlines $\{d'_i\}_{1\leq i\leq n}$ with the precedence-consistent deadlines.

(ii) For each operation O_i, define the backward scheduling problem $B(O_i, S_i)$ with $S_i \stackrel{\text{def}}{=} \mathsf{succ}O_i \cup \mathsf{indep}O_i$.

(1) Let O_i be the current operation in some iteration over $\{O_i\}_{1\leq i\leq n}$.

(2) Compute the optimal backward schedule date σ'_i of O_i by optimal backward scheduling of $B(O_i, S_i)$.

(3) Update the modified deadline of O_i as $d'_i \leftarrow \sigma'_i + 1$.

(4) Update the modified deadlines of each $O_k \in \mathsf{pred}O_i$ with $d'_k \leftarrow \min(d'_k, d'_i - 1 - l^i_k)$.

(5) Go to (1) until a fixpoint of the modified deadlines $\{d'_i\}_{1\leq i\leq n}$ is reached.

In our modified LPPA, we define the *backward scheduling problem* $B(O_i, S_i)$ as the search for a set of dates $\{\sigma'_j\}_{O_j\in\{O_i\}\cup S_i}$ that satisfy:

(a) $\forall O_j \in S_i : O_i \prec O_j \Rightarrow \sigma'_i + 1 + l^j_i \leq \sigma'_j$

(b) $\forall t \in \mathbb{N}, \forall r \in [1, k] : |\{O_j \in \{O_i\} \cup S_i \wedge \tau_j = r \wedge \sigma'_j = t\}| \leq m_r$

(c) $\forall O_j \in \{O_i\} \cup S_i : r^+_j \leq \sigma'_j < d'_j$

Constraints (a) state that only the precedences between O_i and its direct successors are kept in the backward scheduling problem $B(O_i, S_i)$. Constraints (b) are the resources limitations of typed task systems with UET operations. Constraints (c) ensure that operations are backward scheduled within the precedence-consistent release dates and the current modified deadlines. An *optimal backward schedule* for O_i maximizes σ'_i in $B(O_i, S_i)$.

Let $\{r^+_j\}_{1\leq i\leq n}$ be the precedence-consistent release dates and $\{d'_j\}_{1\leq i\leq n}$ be the current modified deadlines. The simplest way to find the optimum backward schedule date of O_i in $B(O_i, S_i)$ is to search for the latest $s \in [r^+_i, d'_i - 1]$ such that the constrained backward scheduling problem $(\sigma'_i = s) \wedge B(O_i, S_i)$ is feasible. Even though each such constrained problem can be solved in polynomial time by reducing to some $\Sigma^k P|r_j; d_j; p_j = 1|-$ over $\{O_i\} \cup S_i$, optimal backward scheduling of $B(O_i, S_i)$ would require pseudo-polynomial time, as there are up to $d'_i - r^+_i$ constrained backward scheduling problems to solve. Please note that a simple dichotomy search for the latest feasible $s \in [r^+_i, d'_i - 1]$ does not work, as $(\sigma'_i = s) \wedge B(O_i, S_i)$ is infeasible does not imply that $(\sigma'_i = s + 1) \wedge B(O_i, S_i)$ is infeasible.

In order to avoid the pseudo-polynomial time complexity of optimal backward scheduling, we rely instead on a procedure with two successive dichotomy searches for feasible relaxations of constrained backward scheduling problems, like in the original LPPA. Describing this procedure requires further definitions. Assume $l^j_i = -\infty$ if $O_i \not\prec O_j$. Given a constrained backward scheduling problem $(\sigma'_i \in [p, q]) \wedge B(O_i, S_i)$, we define a relaxation $\Sigma^k P|\hat{r}_j; \hat{d}_j; p_j = 1|-$ over the operation set $\{O_i\} \cup S_i$ such that:

$$\left\{\begin{array}{rcl} & & \hat{r}_i \stackrel{\text{def}}{=} p \\ & & \hat{d}_i \stackrel{\text{def}}{=} q + 1 \\ O_j \in S_i & \Longrightarrow & \hat{r}_j \stackrel{\text{def}}{=} \max(r^+_j, q + 1 + l^j_i) \\ O_j \in S_i & \Longrightarrow & \hat{d}_j \stackrel{\text{def}}{=} d'_j \end{array}\right.$$

In other words, the precedences from O_i to each direct successor $O_j \in S_i$ are converted into release dates assuming the release date and deadline of O_i respectively equal p and $q + 1$. We call *type 2 relaxation* the resulting scheduling problem $\Sigma^k P|\hat{r}_j; \hat{d}_j; p_j = 1|-$ and *type 1 relaxation* this

[1] Leung, Palem and Pnueli call them "consistent and stable modified deadlines".

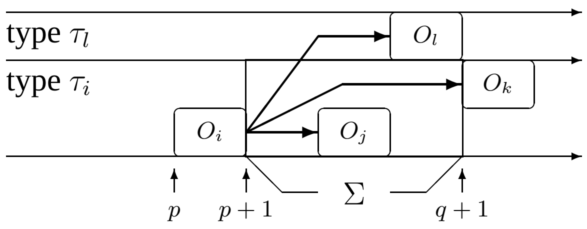


Figure 2: Optimal backward scheduling proof.

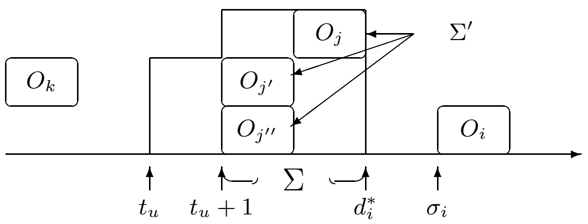


Figure 3: Modified Leung-Palem-Pnueli algorithm proof.

problem when disregarding the resource constraints of O_i. Both type 1 and type 2 relaxations are optimally solved by the GLSA with the earliest $\hat{d}_j$ first priority (Theorem 1). If any relaxation is infeasible, so is the constrained backward scheduling problem $(\sigma'_i \in [p, q]) \wedge B(O_i, S_i)$.

Observe that the type 1 relaxation is increasingly constrained as q increases, independently of the value of p. And for any fixed q, the type 2 relaxation is increasingly constrained as p increases. Therefore, it is correct to explore the feasibility of any of these relaxations using dichotomy search. So the optimal backward scheduling procedure is based on two dichotomy searches as follows.

The first dichotomy search initializes $p = r_i^+$ and $q = d'_i - 1$. Then it proceeds to find the latest q such that the type 1 relaxation is feasible. The second dichotomy search keeps q constant and finds the latest p such that the type 2 relaxation is feasible. Whenever both searches succeed, the optimum backward schedule date of O_i is taken as $\sigma'_i = p$ so the new modified deadline is $d'_i = p + 1$. If any dichotomy search fails, $B(O_i, S_i)$ is assumed infeasible.

Algorithm Proofs

Theorem 2 *The optimal backward scheduling procedure computes the latest schedule date σ'_i of O_i among the schedules that satisfy conditions (a), (b), (c) of $B(O_i, S_i)$.*

Proof: The two dichotomy searches are equivalent to linear searches, respectively by increasing q and by increasing p. If no feasible relaxation $\Sigma^k P|\hat{r}_j; \hat{d}_j; p_j = 1|-$ exist in any of these linear searches, the backward scheduling problem $B(O_i, S_i)$ is obviously infeasible.

If a feasible relaxation exists in the second linear search, this search yields a backward schedule with $\sigma'_i = p$. Indeed, let $\{\hat{\sigma}_j\}_{O_j \in \{O_i\} \cup S_i}$ be schedule dates for the type 2 relaxation of $(\sigma'_i \in [p, q]) \wedge B(O_i, S_i)$. We have $\hat{\sigma}_i = p$ because the type 2 relaxation of problem $(\sigma'_i \in [p+1, q]) \wedge B(O_i, S_i)$ is infeasible and the only difference between these two relaxations is the release date of O_i. Moreover, the dates $\{\hat{\sigma}_j\}_{O_j \in \{O_i\} \cup S_i}$ satisfy (a), (b), (c). Condition (a) is satisfied from the definition of $\hat{r}_j$ and because $\hat{\sigma}_i = p \leq q$. Conditions (b) and (c) are satisfied by the GLSA.

Let us prove that the backward schedule found by the second search is in fact optimal, that is, there is no $s \in [p+1, q]$ such that problem $(\sigma'_i \in [s, s]) \wedge B(O_i, S_i)$ is feasible. This is obvious if $p = q$, so consider cases where $p < q$. The type 2 relaxation of problem $(\sigma'_i \in [p, q]) \wedge B(O_i, S_i)$ is feasible while the type 2 relaxation of problem $(\sigma'_i \in [p + 1, q]) \wedge B(O_i, S_i)$ is infeasible imply there is a set Σ of operations that fill all slots of type τ_i in range $[p + 1, q]$ and prevents the GLSA from scheduling of O_i in that range (Figure 2). So $O_j \in \Sigma \Rightarrow \hat{d}_j \leq \hat{d}_i = q + 1 \wedge \hat{r}_j \geq p + 1$.

Now assume exists some $s \in [p + 1, q]$ such that problem $(\sigma'_i \in [s, s]) \wedge B(O_i, S_i)$ is feasible. This imply that problem $(\sigma'_i \in [p + 1, s]) \wedge B(O_i, S_i)$ is also feasible. The type 2 relaxation of $(\sigma'_i \in [p + 1, s]) \wedge B(O_i, S_i)$ differs from the type 2 relaxation of $(\sigma'_i \in [p+1, q]) \wedge B(O_i, S_i)$ only by the decrease of the release dates $\hat{r}_j$ of some operations $O_j \in S_i$, yet $\hat{r}_j \geq p + 1$ as $\hat{r}_j \overset{\text{def}}{=} \max(r_j^+, s + 1 + l_i^j) \geq p + 1 + 1 + l_i^j$. As all the operations of Σ must still be scheduled in range $[p + 1, q]$ in the type 2 relaxation of $(\sigma'_i \in [p + 1, s]) \wedge B(O_i, S_i)$, there is still no scheduling slot for O_i in that range. So problem $(\sigma'_i \in [p + 1, s]) \wedge B(O_i, S_i)$ and problem $(\sigma'_i \in [s, s]) \wedge B(O_i, S_i)$ are infeasible. □

Theorem 3 *The modified algorithm of Leung, Palem and Pnueli solves any feasible problem $\Sigma^k P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|-$.*

Proof: The correctness of this modified Leung-Palem-Pnueli algorithm (LPPA), like the correctness of the original LPPA, is based on two arguments. The first argument is that the fixpoint modified deadlines are indeed deadlines of the original problem. This is apparent, as each backward scheduling problem $B(O_i, S_i)$ is a relaxation of the original scheduling problem and optimal backward scheduling computes the latest schedule date of O_i within $B(O_i, S_i)$ by Theorem 2. Let us call *core* the GLSA that uses the earliest fixpoint modified deadlines first as priorities. The second correctness argument is a proof that the core GLSA does not miss any fixpoint modified deadlines.

Precisely, assume that some O_i is the earliest operation that misses its fixpoint modified deadline d_i^* in the core GLSA schedule. In a similar way to (Leung, Palem, & Pnueli 2001), we will prove that an earlier operation O_k necessarily misses its fixpoint modified deadline d_k^* in the same schedule. This contradiction ensures that the core GLSA schedule does not miss any fixpoint modified deadline. The details of this proof rely on a few definitions and observations illustrated in Figure 3.

Let $r = \tau_i$ be the type of operation O_i. An operation O_j is said *saturated* if $\tau_j = r$ and $d_j^* \leq d_i^*$. Define $t_u < d_i^*$ as the latest time step that is not filled with saturated operations on the processors of type r. If $t_u < 0$, the problem is infeasible, as there are not enough slots to schedule operations of type r on m_r processors within the deadlines. Else, some scheduling slots of type r at t_u are either empty or filled with operations $O_u : d^*_u > d^*_i$ of lower priority than saturated operations in the core GLSA. Define the operation set $\Sigma \overset{\text{def}}{=} \{O_j \text{ saturated} : t_u < \sigma_j < d^*_i\} \cup \{O_i\}$. Define the operation subset $\Sigma' \overset{\text{def}}{=} \{O_j \in \Sigma : r^+_j \leq t_u\}$.

Consider problem $P^k|intOrder(mono\ l^j_i); r_i; d_i; p_i = 1|-$. In an interval order, given two operations O_i and O_j, either $\mathsf{pred}O_i \subseteq \mathsf{pred}O_j$ or $\mathsf{pred}O_j \subseteq \mathsf{pred}O_i$. Select $O_{j'}$ among $O_j \in \Sigma'$ such that $|\mathsf{pred}O_j|$ is minimal. As $O_{j'} \in \Sigma'$ is not scheduled at date t_u or earlier by the core GLSA, there must be a constraining operation O_k that is a direct predecessor of operation $O_{j'}$ with $\sigma_k + 1 + l^{j'}_k = \sigma_{j'} > t_u \Rightarrow \sigma_k + 1 > t_u - l^{j'}_k$. Note that O_k can have any type. Operations in $\mathsf{pred}O_{j'}$ are the direct predecessors of all operations $O_j \in \Sigma'$ and no predecessor of $O_{j'}$ is in Σ'. Thus $O_k \notin \Sigma'$ and O_k is a direct predecessor of all operations $O_j \in \Sigma'$.

We call *stable backward schedule* any optimal backward schedule of $B(O_k, S_k)$ where the modified deadlines equal the fixpoint modified deadlines. Since $S_k \overset{\text{def}}{=} \mathsf{succ}O_k \cup \mathsf{indep}O_k$, we have $\Sigma \subseteq S_k$. By the fixpoint property, we may assume that a stable backward schedule of $B(O_k, S_k)$ exists. Such stable backward schedule must slot the $m_r(d^*_i - 1 - t_u) + 1$ operations of Σ before d^*_i on m_r processors, so at least one operation $O_j \in \Sigma'$ is scheduled at date t_u or earlier by any stable backward schedule of $B(O_k, S_k)$.

Theorem 2 ensures that optimal backward scheduling of $B(O_k, S_k)$ satisfies the precedence delays between O_k and O_j. Thus $\sigma'_k + 1 + l^j_k \leq t_u$ so $d^*_k - 1 + 1 + l^j_k \leq t_u$. By the monotone interval order property, $\mathsf{pred}O_{j'} \subseteq \mathsf{pred}O_j \Rightarrow w(O_k, O_{j'}) \leq w(O_k, O_j) \Rightarrow 1 + l^{j'}_k \leq 1 + l^j_k \Rightarrow l^{j'}_k \leq l^j_k$ for $O_{j'}$ selected above and $O_j \in \Sigma'$, so $d^*_k \leq t_u - l^{j'}_k$. However in the core GLSA schedule $\sigma_k + 1 > t_u - l^{j'}_k$, so O_k misses its fixpoint modified deadline d^*_k. □

The overall time complexity of this modified LPPA is the sum of the complexity of initialization steps (i-ii), of the number of iterations times the complexity of steps (1-5) and of the complexity of the core GLSA. Leung, Palem and Pnueli (Leung, Palem, & Pnueli 2001) observe that the number of iterations to reach a fixpoint is upper bounded by n^2, a fact that still holds for our modified algorithm. As the time complexity of the GLSA on typed task systems with k types is within a factor k of the time complexity of the GLSA on parallel processors, our modified LPPA has polynomial time complexity.

In their work, Leung, Palem and Pnueli (Leung, Palem, & Pnueli 2001) describe further techniques that enable to lower the overall complexity of their algorithm. The first is a proof that applying optimal backward scheduling in reverse topological order of the operations directly yields the fixpoint modified deadlines. The second is a fast implementation of list scheduling for problems $P|r_i; d_i; p_i = 1|-$. These techniques apply to typed task systems as well.

Table 1: ST200 VLIW processor resource availabilities and operation class resource requirements

Resource	Issue	Memory	Control	Align
Availability	4	1	1	2
ALU	1	0	0	0
ALUX	2	0	0	1
MUL	1	0	0	1
MULX	2	0	0	1
MEM	1	1	0	0
MEMX	2	1	0	1
CTL	1	0	1	1

Application to VLIW Instruction Scheduling

ST200 VLIW Instruction Scheduling Problem

We illustrate VLIW instruction scheduling problems on the ST200 VLIW processor manufactured by STMicroelectronics. The ST200 VLIW processor executes up to 4 operations per time unit with a maximum of one control operation (goto, jump, call, return), one memory operation (load, store, prefetch), and two multiply operations per time unit. All arithmetic operations operate on integer values with operands belonging either to the General Register file (64 × 32-bit) or to the Branch Register file (8 × 1-bit). In order to eliminate some conditional branches, the ST200 VLIW architecture also provides conditional selection instructions. The processing time of any operation is a single time unit ($p_i = 1$), while the precedence delays l^j_i between operations range from -1 to 2 time units.

The resource availabilities of the ST200 VLIW processor and the resource requirements of each operation are displayed in Table 1. The resources are: Issue for the instruction issue width; Memory for the memory access unit; Control for the control unit. An artificial resource Align is also introduced to satisfy some encoding constraints. Operations with identical resource requirements are factored into *classes*: ALU, MUL, MEM and CTL correspond respectively to the arithmetic, multiply, memory and control operations. The classes ALUX, MULX and MEMX represent the operations that require an extended immediate operand. Operations named `LDH`, `MULL`, `ADD`, `CMPNE`, `BRF` belong respectively to classes MEM, MUL, ALU, ALU, CTL.

A sample C program and the corresponding ST200 VLIW processor operations for the inner loop are given in Figure 4. The operations are numbered in their appearance order. In Figure 5, we display the precedence graph between operations of the inner loop of Figure 4 after removing the redundant transitive arcs. As usual in RCPSP, the precedence graph is augmented with dummy nodes O_0 and $O_{n+1} : n = 7$ with null resource requirements. Also, the precedence arcs are labeled with the corresponding start-start time-lag, that is, the values of $p_i + l^i_j$. The critical path of this graph is $O_0 \rightarrow O_1 \rightarrow O_2 \rightarrow O_3 \rightarrow O_7 \rightarrow O_8$ so the makespan is lower bounded by 7.

This example illustrates that null start-start time-lags, or precedence delays $l^i_j = -p_i$, occur frequently in actual VLIW instruction scheduling problems. Moreover, the start-

```
int
prod(int n, short a[], short b) {
   int s=0, i;
   for (i=0;i<n;i++) {
      s += a[i]*b;
   }
   return s;
}
```

```
L?__0_8:
    LDH_1     g131 = 0, G127
    MULL_2    g132 = G126, g131
    ADD_3     G129 = G129, g132
    ADD_4     G128 = G128, 1
    ADD_5     G127 = G127, 2
    CMPNE_6   b135 = G118, G128
    BRF_7     b135, L?__0_8
```

Figure 4: A sample C program and the corresponding ST200 operations

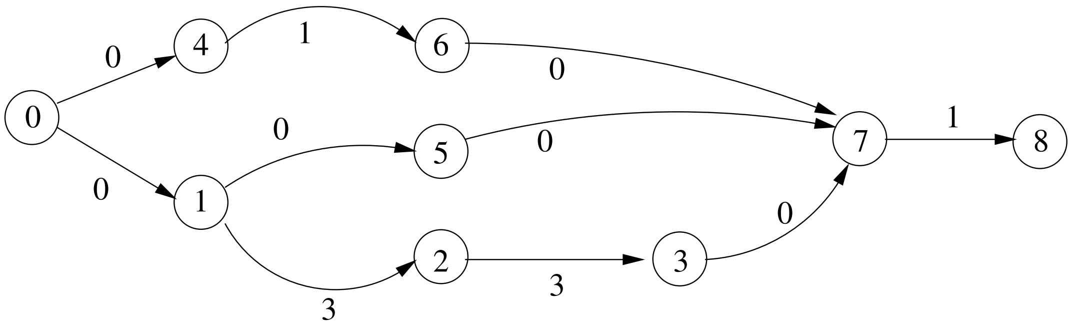


Figure 5: Precedence graph of the inner loop instruction scheduling problem

start time-lags are non-negative, so classic RCPSP schedule generation schemes (Kolisch & Hartmann 1999) (list scheduling) are guaranteed to build feasible (sub-optimal) solutions for these VLIW instruction scheduling problems. In this setting, the main value of VLIW instruction scheduling problem relaxations such as typed task systems is to strengthen the bounds on operation schedule dates including the makespan. Improving bounds benefits scheduling techniques such as solving time-indexed integer linear programming formulations (Dupont de Dinechin 2007).

ST200 VLIW Compiler Experimental Results

We implemented our modified Leung-Palem-Pnueli algorithm in the instruction scheduler of the production compiler for the ST200 VLIW processor family. In order to apply this algorithm, we first relax instances of RCPSP with UET operations and non-negative start-start time-lags to instances of scheduling problems on typed task systems with precedence delays, release dates and deadlines:

- Expand each operation that requires several resources to a chain of sub-operations that use only one resource type per sub-operation. Set the chain precedence delays to -1 (zero start-start time-lags).
- Assign to each sub-operation the release date and deadline of its parent operation.

The result is a UET typed task system with release dates and deadlines, whose precedence graph is arbitrary.

Applying our modified Leung-Palem-Pnueli algorithm to an arbitrary precedence graph implies that optimal scheduling is no longer guaranteed. However, the fixpoint modified deadlines are still deadlines of the UET typed task system considered, as the proof of Theorem 2 does not involve the precedence graph properties. From the way we defined the relaxation to typed task systems, it is apparent that these fixpoint modified deadlines are also deadlines of the original problem (UET RCPSP with non-negative time-lags).

In Table 2, we collect the results of lower bounding the makespan of ST200 VLIW instruction scheduling problems with our modified LPPA for typed task systems. These results are obtained by first computing the fixpoint modified deadlines on the reverse precedence graph, yielding strengthened release dates. The modified LPPA is then applied to the precedence graph with strengthened release dates, and this computes fixpoint modified deadlines including a makespan lower bound. The benchmarks used to extract these results include an image processing program, and the `c-lex` SpecInt program.

The first column of Table 2 identifies the code block that defined the VLIW instruction scheduling problem. Column n gives the number of operations to schedule. Columns Resource, Critical, MLPPA respectively give the makespan lower bound in time units computed with resource use, critical path, and the modified LPPA. The last column ILP gives the optimal makespan as computed by solving a time-indexed linear programming formulation (Dupont de Dinechin 2007). According to this experimental data, there exists cases where using the modified LPPA yields a significantly stronger relaxation than critical path computation.

Summary and Conclusions

We present a modification of the algorithm of Leung, Palem and Pnueli (LPPA) (Leung, Palem, & Pnueli 2001) that schedules monotone interval orders with release dates and deadlines on UET typed task systems (Jaffe 1980) in poly-

Table 2: ST200 VLIW compiler results of the modified Leung-Palem-Pnueli algorithm

Label	n	Resource	Critical	MLPPA	ILP
BB26	41	11	15	19	19
BB23	34	10	14	18	18
BB30	10	3	5	5	5
BB29	16	5	10	10	10
_1_31	34	9	14	18	18
BB9_Short	16	4	10	10	10
BB22	16	4	10	10	10
LAO021	22	6	6	7	7
LAO011	20	6	18	18	18
BB80	14	6	17	17	17
LAO033	41	11	31	32	32
4_1362	23	9	38	38	38
BB916	34	14	30	31	31
4_1181	15	8	18	19	19
4_1180	7	2	9	10	10
4_998	14	4	10	11	11
4_1211	9	2	9	9	9
4_1209	14	7	18	18	18
4_1388	6	2	8	9	9
4_949	13	5	12	13	13
BB740	11	4	13	14	14
LAO0160	17	7	7	11	11

nomial time. In an extended $\alpha|\beta|\gamma$ denotation, this is problem $\Sigma^k P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|-$.

Compared to the original LPPA (Leung, Palem, & Pnueli 2001), our main modifications are: use of the Graham list scheduling algorithm (GLSA) adapted to typed task systems and to zero start-start time-lags; new definition of the backward scheduling problem $B(O_i, S_i)$ that does not involve the transitive successors of operation O_i; core LPPA proof adapted to typed task systems and simplified thanks to the properties of monotone interval orders.

Like the original LPPA, our modified algorithm optimally solves a feasibility problem: after scheduling with the core GLSA, one needs to check if the schedule meets the deadlines. By embedding this algorithm in a dichotomy search for the smallest L_{max} such that the scheduling problem with deadlines $d_i + L_{max}$ is feasible, one also solves $\Sigma^k P|intOrder(mono\ l_i^j); r_i; p_i = 1|L_{max}$ in polynomial time. This is a significant generalization over the $\Sigma^k P|intOrder; p_i = 1|C_{max}$ problem solved by Jansen (Jansen 1994) in polynomial time.

Our motivation for the study of typed task systems with precedence delays is their use as relaxations of the Resource-Constrained Scheduling Problems (RCPSP) with Unit Execution Time (UET) operations and non-negative start-start time-lags. In this setting, precedence delays are important, yet no previous polynomial-time scheduling algorithms for typed task systems consider them. The facts that interval orders include operations without predecessors and successors, and that the LPPA enforces releases dates and deadlines, are also valuable for these relaxations.

References

Ali, H. H., and El-Rewini, H. 1992. Scheduling Interval Ordered Tasks on Multiprocessor Architecture. In *SAC '92: Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing*, 792–797. New York, NY, USA: ACM.

Brucker, P.; Drexl, A.; Möhring, R.; Neumann, K.; and Pesch, E. 1999. Resource-Constrained Project Scheduling: Notation, Classification, Models and Methods. *European Journal of Operational Research* 112:3–41.

Brucker, P. 2004. *Scheduling Algorithms, 4th edition*. SpringerVerlag.

Chaudhuri, S.; Walker, R. A.; and Mitchell, J. E. 1994. Analyzing and Exploiting the Structure of the Constraints in the ILP Approach to the Scheduling Problem. *IEEE Transactions on VLSI* 2(4).

Dupont de Dinechin, B. 2004. From Machine Scheduling to VLIW Instruction Scheduling. *ST Journal of Research* 1(2). http://www.st.com/stonline/press/magazine/stjournal/vol0102/.

Dupont de Dinechin, B. 2007. Time-Indexed Formulations and a Large Neighborhood Search for the Resource-Constrained Modulo Scheduling Problem. In *3rd Multidisciplinary International Scheduling conference: Theory and Applications (MISTA)*. http://www.cri.ensmp.fr/classement/2007.html.

Garey, M. R., and Johnson, D. S. 1976. Scheduling Tasks with Nonuniform Deadlines on Two Processors. *J. ACM* 23(3):461–467.

Jaffe, J. M. 1980. Bounds on the Scheduling of Typed Task Systems. *SIAM J. Comput.* 9(3):541–551.

Jansen, K. 1994. Analysis of Scheduling Problems with Typed Task Systems. *Discrete Applied Mathematics* 52(3):223–232.

Kolisch, R., and Hartmann, S. 1999. Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis. In J., W., ed., *Handbook on Recent Advances in Project Scheduling*. Kluwer Academic. chapter 7.

Leung, A.; Palem, K. V.; and Pnueli, A. 2001. Scheduling Time-Constrained Instructions on Pipelined Processors. *ACM Trans. Program. Lang. Syst.* 23(1):73–103.

Palem, K. V., and Simons, B. B. 1993. Scheduling Time-Critical Instructions on RISC Machines. *ACM Trans. Program. Lang. Syst.* 15(4):632–658.

Papadimitriou, C. H., and Yannakakis, M. 1979. Scheduling Interval-Ordered Tasks. *SIAM J. Comput.* 8(3):405–409.

Verriet, J. 1996. Scheduling Interval Orders with Release Dates and Deadlines. Technical Report UU-CS-1996-12, Department of Information and Computing Sciences, Utrecht University.

Verriet, J. 1998. The Complexity of Scheduling Typed Task Systems with and without Communication Delays. Technical Report UU-CS-1998-26, Department of Information and Computing Sciences, Utrecht University.

A Note on Concurrency and Complexity in Temporal Planning

Maria Fox and Derek Long
Department of Computer & Information Sciences
University of Strathclyde, Glasgow, UK

Abstract

Rintanen recently reported (Rintanen 2007) that the complexity of temporal planning with durative actions of fixed durations in propositional domains depends on whether it is possible for multiple instances of the same action to execute concurrently. In this paper we explore the circumstances in which such a situation might arise and show that the issue is directly connected to previously established results for compilation of conditional effects in propositional planning.

1 Introduction

In his paper *Complexity of Concurrent Temporal Planning* (Rintanen 2007), Jussi Rintanen shows that temporal planning in propositional domains, with durative actions of fixed durations, can be encoded directly in a propositional planning framework by using (propositionally encoded) counters to capture the passage of time. Actions are split into their end points, in much the same way as shown in the semantics of PDDL2.1 (Fox & Long 2003) and as implemented in some planners (Halsey, Long, & Fox 2004; Long & Fox 2003). This encoding allows him to deduce that the complexity of this form of temporal planning is equivalent to that of classical planning when the number of such counters is polynomial in the size of the original (grounded) domain. However, if multiple instances of the same action may execute concurrently then it is not sufficient to have a single counter for each action instance, but instead as many counters are required as potential instances of the same action that may run concurrently. Rintanen observes that this could be exponential in the size of the domain encoding, placing the planning problem into a significantly worse complexity class than classical planning: EXPSPACE-hard instead of PSPACE-hard.

In this paper, we explore the situations in which instances of the same action can run concurrently and link the complexity costs the previously recognised problem of compiling conditional effects into classical propositional encodings (Gazen & Knoblock 1997; Nebel 2000).

2 Preliminaries

We begin by providing some definitions on which the remainder of the paper is based.

Definition 1 *A* classical propositional planning action, *a, is a triple, $\langle P, A, D\rangle$, where each of P, A and D is a set of atomic propositions. The action is applicable in a state, S, also represented by a set of atomic propositions, if $P \subseteq S$. The effect of execution of a will be to transform the state into the new state $a(S) = (S - D) \cup A$.*

Although states are sets of propositions, not all sets of propositions form valid states for a given domain. For a given domain, consisting of an initial state, a collection of actions and a goal condition, the set of states for the domain is the set of all sets of propositions that can be reached by legal applications of the actions. In the rest of the paper, when we quantify over states we intend this to be over all the valid states for the (implicit) domain in question.

Definition 2 *A* simple durative propositional action, *D, with fixed duration (Fox & Long 2003), is the 4-tuple $\langle A_s, A_e, I, d\rangle$, where d is the duration (a fixed rational), A_s and A_e are classical propositional planning actions that define the pre- and post-conditions at the start and end points of D respectively, and I is an invariant condition, which is a set of atomic propositions that must hold in every state throughout the execution of D.*

We do not choose to emphasise the conditions under which two classical actions are considered mutex, here (see (Fox & Long 2003) for details), but note that concurrent execution of two instances of the same durative action in which the end points coincide will not be possible if the end points are mutex. This means that they cannot delete or add the same propositions, so that they actually have no effects. Hence, there is no role for these actions in a plan and they can be ignored in planning. Therefore, we assume that all our durative actions must, if two instances of the same action are to run concurrently, be executed with some offset between them.

3 Key Properties of Actions

We now proceed to define some essential properties that help to characterise the ways in which actions can interact with one another or with aspects of the states to which they are applied.

Definition 3 *A classical propositional action, $a = \langle P, A, D\rangle$ is* repeatable *if in every state S in which a is applicable, $P \subseteq a(S)$.*

A repeatable action can be applied twice in succession without any intervening action to reset the state of resources that might be used by the action. As we shall see, repeatable actions are constrained in the impact they may have on a state.

Definition 4 *A classical propositional action, $a = \langle P, A, D\rangle$ is* weakly conditional *if there are two states S_1 and S_2 such that a is applicable in both states and either there is a proposition $p \in A$ such that $p \in S_1$ and $p \notin S_2$ or there is a proposition $p \in D$ such that $p \in S_1$ and $p \notin S_2$.*

A weakly conditional action is one that can be executed in situations in which some of its positive effects are already true, despite not being preconditions of the action, or some of its negative effects are already false. The reason we call these actions weakly conditional is that these effects are semantically equivalent to the simple conditional effects *(when (not p) p)* and *(when p (not p))* for positive and negative effects respectively. These expressions are obviously part of a richer language than the classical propositional actions. In fact, they make use of both negative preconditions and conditional effects. This combination is known to be an expensive extension to the classical propositional framework (Gazen & Knoblock 1997; Nebel 2000). Nevertheless, weakly conditional actions are obviously valid examples of classical propositional actions. Notice that we require weakly conditional actions to be applicable in states that capture both possibilities in the implicit condition. This constraint ensures that situations in which the preconditions of an action imply that a deleted condition must also hold, without that condition being explicitly listed as a precondition (or the analogous case for an add effect) are not treated as examples of weakly conditional behaviour.

We now define some actions with reduced structural content of one form or another.

Definition 5 *A classical propositional action $a = \langle P, A, D\rangle$ is a* null action *if P, A and D are all empty.*

Definition 6 *A classical propositional action $a = \langle P, A, D\rangle$ is a* null-effect action *if for every state S such that $P \subseteq S$, $S = a(S)$.*

Note that one way in which an action can be a null-effect action is that the action simply reasserts any propositions it deletes and all of its effects are already true in the state to which it is applied. Actions that reassert conditions they delete are not entirely useless, provided they also achieve some other effects that are not already true. Some encodings of the blocks world domain can lead to ground actions that both delete and add the proposition that there is space on the table, simply to avoid having to write special actions to deal with the table. Also observe that null actions are a special case of null-effect actions.

We can now prove a useful property of repeatable actions:

Theorem 1 *Any repeatable action is either a weakly conditional action, a null action or a null-effect action.*

Proof: Suppose a repeatable action, $a = \langle P, A, D\rangle$, is not a null-effect action (and, therefore, not a null action). Then there must be some state in which a can be applied, S_a, such that $a(S_a) \neq S_a$. Since a is repeatable, it must be that $P \subseteq a(S_a) = (S_a - D) \cup A \neq S_a$.

Suppose a is not weakly conditional. Then for every $p \in D$, $p \in S_a$ iff $p \in a(S_s)$ and for every $p \in A$, $p \in S_a$ iff $p \in a(S_a)$. Since $A \subseteq a(S_a)$, the latter implies that $A \subseteq S_a$. The fact that $a(S_a) \neq S_a$ then implies that there is some $p \in D$ such that $p \in S_a$ and $p \notin a(S_a)$. This contradicts our assumption, so a must be weakly conditional. □

We now consider the ways in which these classical actions can appear in certain roles in durative actions.

Definition 7 *A simple durative action, $D = \langle A_s, A_e, I, d\rangle$, is a* pseudo-durative action *if A_e is a null action and I is empty.*

Definition 8 *A simple durative action, $D = \langle A_s, A_e, I, d\rangle$, is a* purely state-preserving action *if $A_e = \langle P, A, D\rangle$ is a null-effect action and every state satisfying I also satisfies P.*

3.1 Deadlocking Actions

One last variety of action is so significant we choose to devote a separate subsection to it.

Definition 9 *A simple durative action, $D = \langle A_s, A_e, I, d\rangle$, is a* deadlocking action *if there is a state, S, such that $I \subseteq S$ but A_e is not applicable in S.*

Thus, a deadlocking action is one that could begin execution and then, either by execution of intervening actions, or possibly simply by leaving

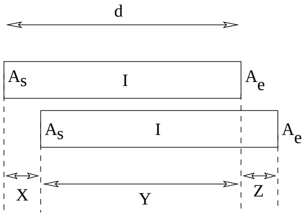


Figure 1: The intervals created by overlapping execution of two instances of the same action.

the state unchanged, it is possible to arrive in a situation in which the action cannot terminate because the conditions for its termination are not satisfied.

Deadlocking actions are clearly no natural actions: there is no real situation in which it is possible to stop time advancing by entering a state in which an action must terminate before time progresses, but cannot because the conditions for its termination are not satisfied. If we adopt a model in which a durative action has a fixed duration then the conditions for its termination must be inevitable, but the effects it has might well be conditional on the state at that time. In domains where deadlock is possible (for example, in the execution of parallel processes), the effect is not to stop time, of course, but to stop execution of the processes. This means that if one were to consider the behaviour of the parallel processes to be modelled by durative actions, the failure to terminate is handled by the actions having unlimited duration.

Therefore, we contend that no natural domains require to be modelled with deadlocking actions.

4 Self-Overlapping Actions

We now turn our attention to the circumstances in which two instances of the same simple durative action can be executed concurrently. Figure 1 shows the intervals that are created by the overlapping execution of such a pair of action instances. Note that when such an overlap occurs there are two places where classical propositional actions might be repeated: A_s and A_e.

Theorem 2 *If two instances of a simple durative action, $a = \langle A_s, A_e, I, d\rangle$ can execute concurrently, then either a is either a deadlocking, pseudo-durative or purely state-preserving action, or else A_e is weakly conditional.*

Proof: Suppose that two instances of a can execute concurrently and consider the two instances of A_e at the ends of the action instances. Either a is deadlocking, or else it must be possible for these two instances to be repeatable, since there is no requirement that an action be inserted in the period Z. Then, by our earlier result, A_e must be either a null action, a null-effect action or else weakly conditional. If A_e is a null action then a is either pseudo-durative (if I is empty) or else it is purely state-preserving. Finally, if a is not deadlocking and A_e is a null-effect action, then any preconditions of A_e must be true in any state satisfying I (otherwise there would be a state in which I was satisfied, yet a could not terminate, implying that a is deadlocking) and therefore a is either pseudo-durative (I is empty) or else it is purely state-preserving. □

Now that we have classified the simple durative actions that may execute concurrently with themselves, we briefly analyse the alternatives. We have already argued that deadlocking actions do not appear in natural domains. Pseudo-durative actions can be treated as though they were classical propositional actions, without duration, provided that a simple check is carried out on completed plans to ensure that adequate time is allowed for any instances of these actions to complete. Purely state-preserving actions are more interesting. An example of such an action is an action that interacts with a savings account that then triggers a constraint that the money in the account must be left untouched for some fixed period. Clearly, such an action is not unreasonable, even if it is uncommon. Fortunately, Rintanen's translation of temporal domains into classical domains can be achieved for purely state-preserving actions without additional counters to monitor the duration of overlapping instances of these actions. This is because the only important thing about these actions is how long the conditions they encapsulate must be preserved. Each time a new instance is executed, the clock must be restarted to ensure that the preservation period continues for the full length of the action from that point. Since the end of the action has no effects it is not necessary to apply it except when the counter reaches zero, at which point the invariant constraint becomes inactive.

Thus, the source of the complexity gap that Rintanen identifies can be traced, for all practical purposes, to the use of durative actions terminated by weakly conditional actions. Weakly conditional actions can be compiled into non-weakly conditional actions by the usual expedient of creating multiple versions of the actions. The idea is to have one version for the case where the condition is true and one for the case where the condition is false, each with the appropriate additional precondition to capture the case and the appropriate version carrying the conditional effect, but now as an unconditional effect. The problem with this compilation is that it causes an exponential number of variants to be created in the size of the collection of conditional effects.

In general, the current collection of benchmark domains do not appear to contain durative actions with repeatable terminating actions (although in many cases this is because the states in which the end actions can be executed are limited by the necessary application of the start effects of the durative actions to which they belong). This means that the problem of self-overlapping actions does not arise in these domains.

In domains in which there are repeatable terminating actions, it is non-trivial to identify which effects contribute to the weakly conditional behaviour. Delete effects are simpler to manage: any delete effect that is not listed as a precondition can be assumed to have the potential to be a weakly conditional effect. Add effects are more problematic: unless an add effect is shown to be mutually exclusive with the preconditions of the action, it must be assumed that it is weakly conditional. It is possible to use mutex inference, such as that used in Graphplan (Blum & Furst 1995) or that performed by TIM (Fox & Long 1998), to identify which add effects must be considered as weakly conditional. In general, to ensure that the weakly conditional behaviour has been completely compiled out, it is necessary to make a conservative assumption about any effects that cannot be shown to be ruled out. Nevertheless, in practical (propositional) domains the number of effects is tightly limited (ADL domains with quantified effects are not quite so amenable) and this makes it possible to compile out the weakly conditional effects with a limited expansion in the number of actions.

5 Relevance to Practical Planning

The relevance to practical planner design of the result we have demonstrated is two-fold. Firstly, we have shown that treatment of overlapping instances of the same action can only occur under limited conditions. These conditions can often be identified automatically using standard domain analysis techniques (Fox & Long 1998). This means that it is possible to determine whether machinery is required to be activated to handle the special case. Avoiding the use of techniques that would be redundant is useful in practical planner design, as a way to achieve improved efficiency.

Secondly, the results demonstrate that the focus of temporal planning should be, in the first place, on handling concurrency between distinct action instances and on the treatment of weakly conditional effects. The latter phenomenon is one that has not, to the best of our knowledge, been highlighted in the past, but is clearly a significant issue, since compilation of such effects into unconditional actions is both non-trivial and also, potentially, exponentially costly.

6 Conclusions

We have shown what kinds of simple durative actions can run concurrently with instances of themselves. Identifying the conditions that allow this has led to the discovery of a close link between the complexity gap identified by Rintanen and the complexity induced by the extension of propositional domains to those with conditional effects. A further important consequence of this analysis is to learn that if actions have bounded effect lists then the complexity of temporal planning is PSPACE-complete, even if self-overlapping actions are allowed.

References

Blum, A., and Furst, M. 1995. Fast planning through plan-graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI95)*, 1636–1642.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *Journal of AI Research* 9.

Fox, M., and Long, D. 2003. PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of AI Research* 20:61–124.

Gazen, B., and Knoblock, C. 1997. Combining the expressivity of UCPOP with the efficiency of Graphplan. In *ECP-97*, 221–233.

Halsey, K.; Long, D.; and Fox, M. 2004. Crikey - a planner looking at the integration of scheduling and planning. In *Proceedings of the Workshop on Integration Scheduling Into Planning at 13th Internati onal Conference on Automated Planning and Scheduling (ICAPS'03)*, 46–52.

Long, D., and Fox, M. 2003. Exploiting a graphplan framework in temporal planning. In *Proceedings of ICAPS'03*.

Nebel, B. 2000. On the expressive power of planning formalisms: Conditional effects and boolean preconditions in the STRIPS formalism. In Minker, J., ed., *Logic-Based Artificial Intelligence*. Kluwer. 469–490.

Rintanen, J. 2007. Complexity of concurrent temporal planning. In *Proceedings of International Conference on Automated Planning and Scheduling*, 280–287.

Optimisation of Generalised Policies via Evolutionary Computation

Michelle Galea and **John Levine** and **Henrik Westerberg**[*]
Department of Computer & Information Sciences
University of Strathclyde
Glasgow G1 1XH, UK

Dave Humphreys[†]
CISA, School of Informatics
University of Edinburgh
Edinburgh EH8 9LE, UK

Abstract

This paper investigates the application of Evolutionary Computation to the induction of generalised policies. A policy is here defined as a list of rules that specify which actions to be performed under which conditions. A policy is domain-specific and is used in conjunction with an inference mechanism (to decide which rule to apply) to formulate plans for problems within that domain. Evolutionary Computation is concerned with the design and application of stochastic population-based iterative methods inspired by natural evolution. This work illustrates how it may be applied to the induction of policies, compares the results on one domain with those obtained by a state-of-the-art approximate policy iteration approach, and highlights both the current limitations (such as a simplistic knowledge representation) and the advantages (including optimisation of rule order within a policy) of our system.

Introduction

We present an evolution-inspired system that induces generalised policies from available solutions to planning problems. The term generalised policy was coined by Martin & Geffner (2004) for a function that maps pairs of initial and goal states to actions. The actions outputted should, when performed, achieve the specified goal state from the specified initial state.

Figure 1 presents a simplified view of a planner based on generalised policies. A distinction is made here between a policy – the knowledge used to solve a problem, and the inference mechanism that utilises the policy – the decision procedure that dictates when and how the knowledge is applied. A domain model defines a specific domain in terms of relevant objects, actions and their effects.

A policy in this work is a list of domain-specific IF-THEN rules. If the conditions stated in the IF- part of a rule match the current state, then the action in the THEN part *may* be applied. The currently implemented inference mechanism is a common and simple one – rules within a policy are ordered and the action of the first rule that may be applied is performed. If more than one valid combination of variable bindings exists then orderings on the variables and their values are adopted and the first valid combination is effected.

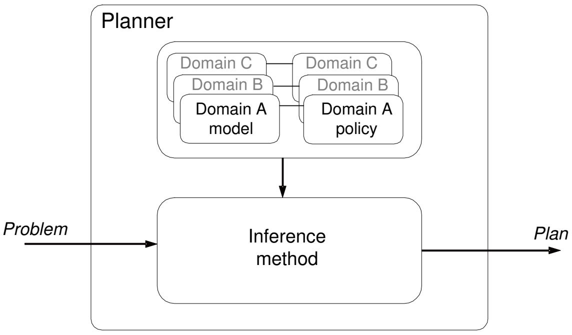


Figure 1: Planning using generalised policies and inference mechanisms

It should be noted that these policies contain a particular type of control knowledge. Control knowledge is domain-specific knowledge often used by some planners to prune search during the construction or identification of a plan. Control knowledge is often expressed as IF-THEN type rules, but the conditions and actions relate to goal, domain operator and/or variable binding decisions to be taken during the search process. Examples of work that induce such knowledge include (Leckie & Zukerman 1998) and (Aler, Borrajo, & Isasi 2002).

In this work a policy determines domain operator selection and each rule describes the conditions necessary for a particular operator to be applied. The inference mechanism is responsible for deciding all other decisions (which rule to apply, and which variable bindings to implement) *without* recourse to any search, leading to highly efficient planners.

The induction of policies is carried out using Evolutionary Computation (EC) in a supervised learning context. EC is the application of methods inspired by Darwinian principles of evolution to computationally difficult problems, such as search and combinatorial optimisation. Its popularity is due in great part to its parallel development and modification of multiple solutions in diverse areas of the solution space, discouraging convergence to a suboptimal solution.

We compare the performance of one evolved policy with that obtained using a state-of-the-art Approximate Policy Iteration (API) (Bertsekas & Tsitsiklis 1996) approach. We

[*] Currently at Systems Biology Unit, Centre for Genomic Regulation, C/Dr Aiguader 88, Barcelona 08003, Spain

[†] Currently at Mobile Detect Inc., Ottawa, Ontario, Canada K1L 6P5

focus on the knowledge representation language (KR) and learning mechanism highlighting both the current limitations and strengths of our system. The rest of this paper reviews the literature on generalised policy induction, describes our implemented system, and discusses experiment results and future research directions.

Related Work

Early work on inducing generalised policies utilises genetic programming (GP) (Koza 1992), a particular branch of EC. Evolutionary algorithms in general re-iteratively apply genetic-inspired operators to a population of solutions, with fitter individuals of a generation (according to some predefined fitness criteria) more likely to be selected for modification and insertion into successive generations than weaker members. On average, therefore, each new generation tends to be fitter than the previous one. GP is distinguished by a tree representation of individuals that makes it a natural candidate for the representation of functional programs.

Koza (1992) describes a GP algorithm for solving a blocksworld problem variant – the goal is a program capable of producing a tower of blocks that spells "UNIVERSAL", starting from a range of different initial tower configurations. The tree-like individuals in a generation are constructed from sets of *functions* (such as `move_to_stack` and `move_to_table`) and *terminals* that act as arguments to the functions (such as `top_block_of_stack` and `next_needed_block`).

Each individual in the population is assessed by its performance on a set of 166 initial configurations. Generation 10 produces a program that correctly stacks the tower for each of the given configurations, though it uses unnecessary block movements and contains unnecessary functions. When elements are included in the fitness assessment that penalise against these inefficiencies, the algorithm outputs a parsimonious program that produces solutions that are both correct and optimal (in terms of plan length).

Spector (1994) uses Koza's algorithm with different function and terminal sets to induce solutions to the Sussman Anomaly – the initial state is block C on block A, with blocks A and B on the table; the goal state is block A on B, which is on C, which is on the table. In a first experiment the author uses functions such as `newtower` (move X to table if X is clear) and `puton` (put X on Y if both are clear), and the terminals are the names of the blocks A, B and C. The goal is a program that can attain the goal state from the initial state, and individuals are assessed on this one problem. The fitness function includes elements that reward parsimony and efficiency as well as correctness, and the goal is achieved well before the final generation.

In further experiments the author introduces new functions and replaces the block-specific terminals with ones that refer to blocks by their positions in goals. The number of problems on which individuals are assessed is also increased. One experiment is designed to produce a program that achieves the Sussman goal state from a range of different initial states. The resulting program achieves this particular goal state even from initial configurations that are not used during learning. However, it is incapable of achieving a different goal state from that on which it was trained, even a simplified one such as (ON B A).

Another experiment seeks a program capable of achieving 4 different goals states (maximum 3 blocks), from different initial states. This evolved program is capable of attaining any of the 4 specified goal states from initial states not observed during the evolutionary process. The author indicates that it is also capable of solving some 4-block problems, though its generalisation power for this and larger problems has not been fully analysed.

The work of Khardon (1999) for inducing policies has inspired and/or often been cited by later work. It uses a deterministic learning method to induce decision lists of IF-THEN rules from examples of solved problems, with the first rule in the list that matches the observed state being applied. The learning strategy is one of iterative rule learning where the following step is iterated until no examples are left in the training set – a number of rules are generated, the best (according to some criterion) is determined, examples that are covered by this rule are removed from the training data, and the rule is added to a growing rulebase. The number of rules generated in each iteration must be finite and tractable and this is controlled in part by setting limits to the number of conditions and variables in the IF- part of a rule; all possible rules for each action are then generated in each iteration. The training data is formulated by extracting examples from planning problems and their solutions – each state and action encountered in a plan constitutes one example.

In addition to the training examples and a standard STRIPS domain description Khardon provides the learning algorithm with background knowledge he calls *support predicates* – concepts such as `above` and `inplace` for the blocksworld domain. The resulting policy is an ordered list of existentially quantified rules with predicates in the condition part that may or may not be negated, and may or may not refer to a subgoal. For instance, $holding(x_1)\ \neg clear(x_2)\ G(on(x_1, x_2)) \rightarrow \textsc{putdown}(x_1)$, represents a rule that says if x_1 is currently held, x_2 is not clear, and in the goal state x_1 should be on x_2, put down x_1.

Blocksworld policies are generated using different training sets containing examples drawn from solutions to 8-block problems, and are tested on new problems of sizes ranging 7–20 blocks. Their performance varies from a high of 83% of 7-block problems solved, down to 56% of 20-block problems. Similar experiments are carried out for the logistics domain with training of policies on examples obtained from solutions to problems with 2 packages, 3 cities, 3 trucks, 2 locations per city, and 2 airplanes. Polices are tested on problems with similar dimensions to the testing problems, and the number of packages is varied from 2, solving 80% of problems, to 30, solving 68% of problems.

Martin & Geffner (2004) suggest that the generalisation power of Khardon's policies over large problems is weak, and that obtaining domain-dependent background knowledge is not always a trivial task. They use the same learning method as Khardon but propose to overcome both weaknesses by using description logics (Baader *et al.* 2003) as the KR. This enables the representation of concepts that describe *classes* of objects, such as the concept of a well-placed block.

A blocksworld policy induced from 5-block problem examples solves 99% of the 25-block test problems. With the addition of an incremental refinement procedure a policy is eventually induced that solves 100% of test problems: a policy is induced and tested on 5-block problems; optimal solutions are found for the problems it fails on, and examples are extracted from these and added to the training set; then, a new policy is induced from the larger dataset. The authors repeat this procedure several times until a policy solves all the 25-block test problems presented (test problems are new each time the policy is tested). It should be noted however that as well as the KR and the refinement extension to the learning algorithm, the way training examples are extracted from solutions is different from that in Khardon's work – Martin & Geffner use as examples *all* actions for each state that lead to an optimal plan; this may have some impact on the quality of the induced policies.

Fern, Yoon, & Givan (2006) learn policies for a long random walk (LRW) problem distribution using a form of API. A policy is a list of action-selection rules where the action of the first rule that matches the current and goal states is applied. An LRW distribution randomly generates an initial state for a problem, executes a long sequence of random actions, and sets the goal as a subset of properties of the final resulting state. For a given domain API iteratively improves a policy until no further improvement is observed or some other stopping criterion is used. The expectation is that if a learned policy π_n performs well on problems drawn from random walks of length n, then it will provide reasonable performance or guidance on problems drawn from random walks of length m, where m is only moderately larger than n. π_n is therefore used to bootstrap API iterations to find π_m, i.e. to find a policy that handles problems drawn from increasingly longer random walks.

Within each iteration, trajectories (sequences of alternating states and actions) for an improved policy are generated using policy rollout (Tesauro & Galperin 1996), and then an improved policy is learned using the trajectories as training data. The policy learning component follows an iterative rule learning strategy. The difference between this learning strategy and that of Khardon and Martin & Geffner lies in the rule generation procedure where a greedy heuristic search is used instead of exhaustively enumerating all rules. The KR (based on taxonomic syntax) is also different, and is expressive enough so that no support predicates need be supplied to the learning process.

This work is currently state-of-the-art in this particular research area, i.e. where policies that are learned are used with a simple and efficient decision procedure to solve planning problems. It presents policies for several domains and tests them rigorously on deterministic and stochastic problems from an LRW distribution and from the 2000 planning competition; the results compare favourably with those obtained by the *FF* planning system (Hoffmann & Nebel 2001).

In this paper we explore the Briefcase domain API-generated policy and compare its performance with one evolved by our system, focusing on the limitations of our KR and the strength of our policy optimisation mechanism.

```
(1) Create initial population
(2) WHILE termination criterion false
(3)   Evaluate current generation
(4)   WHILE new generation not full
(5)     Perform reproduction
(6)     Perform recombination
(7)     Perform mutation
(8)     Perform local search
(9)   ENDWHILE
(10) ENDWHILE
(11) Output fittest individual
```

Figure 2: Pseudocode outline of *L2Plan*

Learning Policies using *L2Plan*

L2Plan (Learn to Plan) induces policies of rules similar to Khardon's, but the learning mechanism used is a population-based iterative approach inspired by natural evolution.

Input to *L2Plan* consists of an untyped STRIPS domain description, additional domain knowledge if available (e.g. concept of a well-placed block), and domain examples on which to evaluate the policies being learned. The output is a domain-specific policy that is used in conjunction with an inference mechanism to solve problems within that domain.

A policy consists of a list of rules with each rule being a specialised IF-THEN rule (also known as a production rule). The IF- part is composed of two condition statements where each is a conjunction of ungrounded predicates which may be negated:

`IF condition AND goalCondition THEN action`

`condition` relates to the current state and `goalCondition` to the goal state. If variable bindings exist such that predicates in `condition` match with the current state, and predicates in `goalCondition` match with the goal state, then the action may be performed. Note though that the action's precondition must also be satisfied in the current state. The list of rules is ordered and the first applicable rule is used. Variable and domain orderings are followed if more than one combination of bindings is possible.

Figure 2 presents an outline of the system. Each iteration starts with a population of policies (line(2)). The performance of these policies is evaluated on training data generated from planning problems from the domain under consideration (line (3)). The resulting measure of fitness for a policy is used to determine whether it is replicated in the next iteration (line (5)), or whether it may be used in combination with another policy to reproduce 'offspring' that may be inserted into the next iteration (also called crossover, line (6)). All policies to be inserted in the next iteration may undergo some form of random mutation (i.e. small change, line (7)), and a local search procedure that attempts to increase the fitness of the policy (line (8)).

The system terminates if a predefined maximum number of generations have been created, or a policy attains maximum fitness by correctly solving all examples, or, the average difference in policy fitness in an iteration falls below a predefined user-set threshold (indicating convergence of all individuals to similar policies).

Since the results of the evaluation process influence the creation of the next generation, the average fitness of all policies is expected to improve from one generation to the next. The fact that several policies are in each iteration allows the

```
(:rule position_briefcase_to_pickup_misplaced_object
  :condition (and (at ?obj ?to))
  :goalCondition (and (not(at ?obj ?to)))
  :action movebriefcase ?bc ?from ?to)
```

Figure 3: Example of a briefcase rule with a variable in `condition` that is not a parameter of the action

possibility of exploring different regions of the solution space at once. This, coupled with an element of randomness that is used in the selection of policies crossover and mutation, may help to prevent all policies from converging to a local optimum solution.

The following paragraphs describe the creation of the initial population, policy evaluation, and the genetic operators used to create new policies from old.

Generating the Initial Population

L2Plan first generates an initial – the first generation – population of policies, Fig. 2 line (1). The number of individuals in a population is predefined by the user (generally 100), and stays fixed until the system terminates. The number of rules in a policy at this stage is randomly set between user-defined minimum and maximum values (4 and 8 respectively).

The `condition` and `goalCondition` statements of a rule are also generated randomly, within certain constraints. The action, i.e. the THEN part of the IF-THEN rule, is first selected randomly from all domain actions.

The size of `goalCondition` in the IF- part of the rule is determined by drawing a random integer between user-defined minimum and maximum values (set to 1 and 3 respectively), which determines the number of predicates. A predicate is first selected, and then the appropriate number of variables are randomly selected from all possible variables. Predicates are randomly negated.

The size of `condition` in the IF- part of the rule is currently determined by the number of parameters of the selected action, and a random selection of predicates. A predicate is selected randomly, and then variables for the predicate are randomly selected from the action's parameters. Predicates are selected, and variables assigned, until all of an action's parameters are present in at least one predicate of `condition`. Each predicate is randomly negated.

However, early experiments highlighted that restricting the parameters in `condition` strictly to those in the set of parameters for an action, severly limits the knowledge that can be expressed by a rule. For example, the system is unable to learn the rule in Fig. 3 due to this constraint. This rule specifies that if an object is misplaced (i.e. its current location is not the location specified for it in the goal state), then a briefcase is moved to the current location of the object. A temporary quickfix has been implemented that inserts an extra unary predicate in the domain description. With this predicate added to the precondition of each action/operator, it allows *L2Plan* the possibility of creating rules such as the one in Fig. 3.

Note, that a policy need not contain a rule to describe each action in the domain, and that the initially set number of rules for a policy, and the number of predicates in the conditions of a rule is liable to change with the application of genetic operators.

```
(define (example blocks1_1)
(:domain blocksworld)
(:objects 5 4 3 2 1)
(:initial   ...  )
(:goal   ...  )
(:actions
  (move-b-to-b 1 3 4) 1
  (move-b-to-b 1 3 5) 1
  (move-b-to-b 4 2 1) 1
  (move-b-to-b 4 2 5) 1
  (move-b-to-t 1 3) 0
  (move-b-to-t 4 2) 0
  (move-t-to-b 5 1) 2
  (move-t-to-b 5 1) 2) )
```

Figure 4: A training example generated from a blocksworld problem

Evaluating a Policy

The training data on which a policy is evaluated is composed of a number of examples that are generated from a number of planning problems. Each example consists of a state encountered on an optimal plan for the problem from which it is extracted, and a number of actions which may be taken from that state, each with an associated cost.

Consider a planning problem that includes an initial state S_I and a goal state S_G. Each possible action that may be taken from S_I is performed, leading to new states. For each new state a solution that attains S_G is found using an available planner. The length of each solution is determined, and the smallest-size solution is deemed the optimal plan. A cost is now attached to each action performed from S_I: the action that leads to the optimal plan is given a cost of zero, and all other actions are given a cost that is the difference between the length of the solution that they form a part of, and the length of the optimal plan. This now forms one training example on which an evolving policy may be evaluated. Figure 4 shows the representation used for a training example, which is consistent, as far as possible, with STRIPS syntax.

For each state on the optimal plan just determineds the same procedure is followed as for S_I, i.e. all possible actions from the next state on the optimal plan, say S_n, are performed, solutions for each of the resulting states are found, and costs for each possible action taken from S_n are determined from the solutions' length. Each training problem therefore yields as many examples as there are states encountered on the optimal plan. Duplicate training examples are removed so as not to bias *L2Plan* towards any particular scenario(s).

The planner used to generate training examples, i.e. when determining plans to S_G from any state S_n, is a simple one using breadth-first search. This ensures that an optimal plan is obtained and that actions in examples designated as optimal are in fact actions for states encountered on some plan of minimal length. For some domains (e.g. blocksworld and briefcase), in order to speed up the generation of examples hand-coded control rules to prune branches from the search are used; these control rules are designed to ensure that an optimal plan is still determined.

The fitness of a policy is determined by averaging its performance over all examples, where for each example presented it is scored based on whether the selected action forms part of an optimal plan or not. Formula (1) below describes the fitness function where m is the number of training examples and $actionCost_i$ is the cost of the action taken by the policy for training example i:

$$fitness = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{1\ +\ actionCost_i} \qquad (1)$$

Creating a New Generation of Policies

Current *L2Plan* settings are such that the individuals comprising the fittest 5% of a generation are reproduced, improved by a local search mechanism, and then inserted into the next generation. The remainder of the next generation is populated by individuals selected from the current generation and on which various genetic operations are performed. The fitter individuals in the current population have a greater chance of being selected for recombination and mutation, in the expectation that their offspring and/or mutations result in even fitter individuals. However, randomness plays a part in their selection and in the application of genetic operators in an attempt to search different areas of the solution space and to avoid local minima.

Selection of two individuals is performed using tournament selection with a size of 2 (Miller & Goldberg 1995). Crossover or mutation is then applied with some predefined probability (0.9 for crossover, 0.1 for mutation). The output of these operators is a single policy – for crossover the fittest of parents and offspring, and for mutation the fittest of the original policy or mutants. Local search is performed on this policy before it is inserted into the new generation. This procedure is repeated until the new generation is full.

There are three types of crossover that may be performed on the 2 selected policies, and 4 types of mutation that may be performed on the first selected policy:

Single Point Rule Level Crossover A crossover point is randomly chosen in each of the 2 policies, with valid points being before any of the rules (points need not be the same in the 2 policies). Two offspring policies are then created by merging part of the policy of one parent (as delineated by the crossover point), with a part of the other parent (the first part of parent A with the second part of parent B, and the second part of parent A with the first part of parent B).

Single Rule Swap Crossover A randomly selected rule from policy A is swapped with a randomly selected rule from policy B, resulting in two new policies. The replacing rule is inserted in the same position in the policy as the one it is replacing.

Similar Action Rule Crossover Two rules with the same action are randomly selected from the parent policies, one from each. Two new rules are created from the selected rules, one by using `condition` from the first selected rule and `goalCondition` from the second, and the other new rule is created by using `goalCondition` from the first selected rule and `condition` from the second. Each of the two newly created rules replaces the original rule in each of the two parent policies, resulting in 4 new policies.

Rule Addition Mutation A new rule is generated and inserted at a random position in the policy.

Rule Deletion Mutation A randomly selected rule is removed from the policy (if the policy contains more than one rule).

Rule Swap Mutation Two randomly selected rules have their position swapped in the policy (if the policy has more than one rule).

Rule Condition Mutation A randomly selected rule has its `condition` and/or `goalCondition` statement mutated, by replacing the condition statement with a newly generated one, or by removing a predicate from the statement, or by adding a new predicate.

The local search procedure currently used is aimed at increasing the fitness of a policy as quickly as possible. It performs rule condition mutations a predefined number of times (called the local search branching factor). The fittest mutant replaces the original policy, and again, rule condition mutations are performed on the new policy the same predefined number of times. This process is repeated until either no improvement in fitness is exhibited by any mutant over their originator policy, or for a preset maximum number of times (called the local search depth factor).

A Comparison of Two Policies

This study focusses on comparing two policies for the briefcase domain, one generated by *L2Plan* and the other by the API approach introduced in the *Related Work* section (Fern, Yoon, & Givan 2006). The comparison serves two purposes:

- it highlights a current limitation of *L2Plan*, which is the limited expressiveness of the KR; and,
- demonstrates the advantage offered by its policy discovery mechanism, which optimises the rule order in a policy.

The Briefcase domain is chosen partly because it is as yet one of the few domains for which we have evolved *L2Plan* policies, and partly because the knowledge expressed in the API induced policy is such that can be expressed as IF-THEN rules.

The API Policy

Figure 5 presents the briefcase domain policy induced by the API algorithm. A policy provides a mapping from states to actions for a specific domain and consists of a decision list of 'action-selection rules' of the form $a(x_1, ..., x_k)\ :\ L_1, L_2, ...L_m$ where a is a k-argument action type, x_i an action argument variable and L_i is a literal. An API policy is utilised in the same way as an *L2Plan* policy. Each rule describes the action to be taken if a variable binding exists for the rule that matches both the current state and the goal. The current state must also satisfy the preconditions of the action specified by the rule. The rules in a policy are ordered and the rule that is applied in a state is the first rule for which a valid variable binding exists. A lexicographic ordering is imposed on objects in a problem to deal with situations where more than one variable binding for the same rule may be possible.

Below is a simpler example policy for illustrating the main features of the KR used. It is a policy for a blocksworld domain where the goal in all problems is to make all red blocks clear is:

1. putdown(x_1) : $x_1 \in holding$
2. pickup(x_1) : $x_1 \in clear,\ x_1 \in (on^*(on\ red))$

1. PUT-IN: $X_1 \in (GAT^{-1}\ (NOT\ IS - AT)))$
2. MOVE: $(X_2 \in (AT\ (NOT\ (CAT^{-1}\ LOCATION)))) \ \wedge$ $(X_2 \in (NOT\ (AT\ (GAT^{-1}\ CIS - AT))))$
3. MOVE: $(X_2 \in (GAT\ IN)) \ \wedge \ (X_1 \in (NOT\ (CAT\ \ IN)))$
4. TAKE-OUT: $(X_1 \in (CAT^{-1}\ IS - AT))$
5. MOVE: $(X_2 \in GIS - AT)$
6. MOVE: $(X_2 \in (AT\ (GAT^{-1}\ CIS - AT)))$
7. PUT-IN: $(X_1 \in UNIVERSAL)$

Figure 5: API briefcase policy in taxonomic syntax

1. PUT-IN misplaced package in briefcase
2. MOVE briefcase to pickup misplaced package, if briefcase is at its goal location and package does not have same goal location as briefcase
3. MOVE to goal location of package in briefcase, if there is no package in briefcase whose goal location is the same as the current location of briefcase
4. TAKE-OUT package that has arrived at its goal location
5. MOVE briefcase to its goal location
6. MOVE to pickup misplaced package, if briefcase is at its goal location and package has same goal location as briefcase
7. PUT-IN package in briefcase.

Figure 6: API briefcase policy in common language

The primitive classes (unary predicates) in this domain are red, $clear$, and $holding$, while on is a primitive relation (binary predicate). If a domain contains predicates of greater arity, these are converted to equivalent multiple binary predicates. A prefix of g indicates a predicate in the goal state (e.g. $gclear$), while a comparison predicate c indicates that a predicate is true in both the current state and the goal (e.g. $cclear$). A primitive class (relation) is a current-state predicate, goal predicate or comparison predicate, and it is interpreted as the set of objects for which the class (relation) is true in a state s. Compound expressions are formed by the 'nesting' of classes/relations, and/or the application of additional language features such as R^* indicating a chain of a relation R. Expressions have a depth associated with them so that, for intstance, the first expression in rule 2 above has depth 1 and the second expression has depth 3.

Figure 6 is a translation of the policy in Fig. 5 into common language. Upon inspection it is clear that there is potential in this policy to perform unnecessary steps. For instance, rule 2 moves the briefcase away from its current location without first depositing any packages it contains that have as a goal location the current briefcase location. Furthermore, two of the four MOVE rules have as a necessary condition that the briefcase must be at its goal location – this can cause problems and is discussed later on.

This API policy is translated into *L2Plan*-style IF-THEN rules and tested using our implemented inference mechanism on the same problems as our evolved policy. However, it is important to note differences in the KR which highlight the limited expressiveness of our current formulation of IF-THEN rules. Consider rule 3 in Fig. 6 – it states that the briefcase is moved to a goal location of a package within it, *only* if there are NO other packages in the briefcase whose goal locations are the same as the current location of the briefcase. If this is so, then rule 4 is fired instead of rule 3, i.e. packages at their goal location are taken out of the briefcase before the briefcase is moved, despite the order and actions suggested by these two rules.

1. TAKE-OUT package that has arrived at its goal location
2. PUT-IN misplaced package in briefcase
3. MOVE briefcase to pickup misplaced package
4. MOVE to goal location of package in briefcase
5. MOVE briefcase to its goal location

Figure 7: *L2plan* briefcase policy in common language

Table 1: *L2Plan* parameter settings

Parameter	Setting
Range of initial policy size	[4–8]
Population size	100
Maximum number of generations	100
Proportion of policies reproduced	5%
Crossover probability	0.9
Mutation probability	0.1
Local search branching	10
Local search depth	10
Tournament selection size	2

As yet we cannot write rule 3 in *L2Plan*-style rules. This limitation is partly due to the fact that we can only specify individual packages using this KR and not sets of packages. However, if we simplify the API policy's rule 3 and switch the order of the simplified rule 3 with rule 4, then we obtain an equivalent policy we can test and compare with *L2Plan*'s policy. The new rule 3 states: TAKE-OUT package that is at its goal location, and the new rule 4 is: MOVE to goal location of package in briefcase.

The *L2Plan* Policy

Figure 7 presents the *L2Plan* evolved policy against which the API policy is compared. Note that the first four rules are equivalent to the hand-coded control policy introduced in (Pednault 1987) and which is used to prune search for this domain by the *TLPlan* system (Bacchus & Kabanza 2000).

To produce this policy *L2Plan* was run 15 times with identical parameter settings (Table 1) though each time the training examples were generated from 30 different randomly generated problems and their solutions. The training problem complexity is however the same: 5 cities, 2 objects and 1 briefcase. Using different training data for different experiments gives some indication of the impact of different examples on the induced policies, though it should be noted that the element of randomness used in solution construction will also have some influence.

Three of the 15 policies solve all test problems presented (i.e. problems different from the ones used for training), and the policy in Fig. 7 was selected from one of these three. Note that though additional domain knowledge other than the standard STRIPS description may used for inducing a policy,

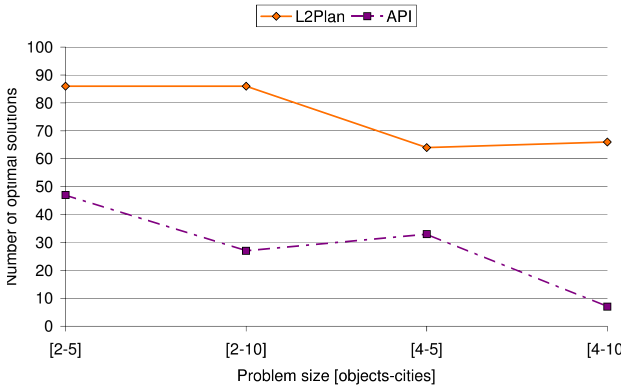


Figure 8: Number of optimal plans produced by a policy

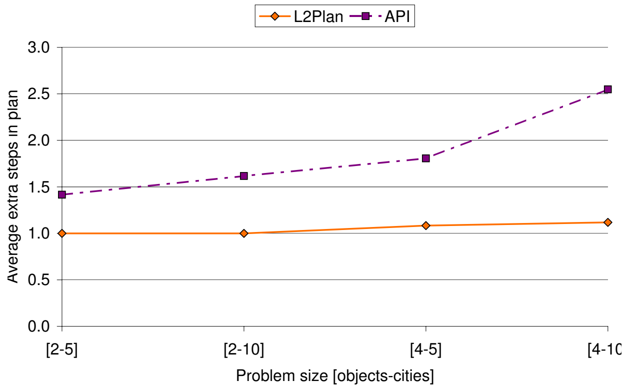


Figure 9: Average number of extra steps in suboptimal plans

none was used during the induction of briefcase policies. Furthermore, little system parameter tuning has been done at this stage, and the settings in Table 1 appear to provide reasonable policies for evolving both briefcase and blocksworld policies (to be discussed briefly later).

Results

Both the API and *L2Plan* policies are run on the same 400 test problems with 1 briefcase: 100 problems each with 2 objects and 5 cities, 2 objects and 10 cities, 4 objects and 5 cities, and 4 objects and 10 cities. These test problems all contain a goal location for the briefcase.

Each policy solves all 400 problems. Figure 8 however depicts the number of problems that a policy manages to solve optimally, i.e. where the plan produced by the policy is no longer than a known optimal plan. Figure 9 shows the average number of extra steps produced per plan by each policy for the problems that were solved suboptimally (i.e. the total of extra steps over all 400 solutions is divided by only the number of suboptimally solved solutins). In both respects the *L2Plan* policy considerably outperforms the API policy – it finds more optimal solutions for problems and generates shorter plans than the API policy when a suboptimal solution is found.

These results are a consequence of the rule order in the respective policies. The API policy moves the briefcase away from its current location without first checking whether an object inside it might be deposited in the current location. *L2Plan* uses several of the crossover and mutation operations to optimise rule order so that the policy is evolved such that it does the most it can do in the current briefcase location – pickups misplaced objects or deposits ones arrived at their current location – before the briefcase is moved.

The API policy also exhibits an apparent dependency on the goal location of the briefcase with several rules checking its location before an action may be taken. To confirm this dependency both policies are run on a new suite of 400 test problems, with the same complexity as the previous suite but without a goal location for the briefcase. Table 2 gives the results achieved by each policy – it shows the total number of problems solved for each problem type, with the number of problems solved optimally (out of the total given) in brackets.

Table 2: Performance of briefcase policies on problems without a goal location for the briefcase. (Number of optimal plans found in brackets)

	Problem size [objects-cities]			
	[2-5]	[2-10]	[4-5]	[4-10]
L2Plan	100 (93)	100 (94)	100 (72)	100 (74)
API	10 (10)	4 (4)	13 (11)	4 (4)

The *L2Plan* policy again solves all 400 problems with a high proportion of them solved optimally.

The performance of the API policy on this suite of problems is however quite different – only a small number of problems are solved, though most of these are solved optimally. This behaviour is a direct consequence of the requirement placed on two of its MOVE rules that the briefcase should be at its goal location before it may be moved. If the briefcase is not at its goal location and no other action can be taken, then rule 5 in this policy moves the briefcase to its goal location and other actions then become possible. The type of problems that this policy has a chance of solving are those where the briefcase *starts out* by being in the same location as one or more of the misplaced packages. The policy dictates that the misplaced packages are put in the briefcase (rule 1), the briefcase is moved to the goal location of one of the packages (rule 3), and the package deposited (rule 4). The policy again dictates that any misplaced packages are picked up from this new location, and again the briefcase is moved to the goal location of a package inside it. However, if the briefcase ends up at some location empty after having delivered a misplaced package, and there are still misplaced packages in other locations then no further action will be possible (since there is no goal location in the problem to which the briefcase can be taken by rule 5).

The *L2Plan* policy has evolved such that the briefcase is moved to its goal location only when all objects have been deposited at their own goal locations (rule 5), and no other rule is dependent on the location of the briefcase.

Conclusions and Future Work

This work suggests that EC is a viable approach for learning generalised policies, and highlights both the limitations and strengths of the current implementation.

IF-THEN rules are a highly comprehensible but also a simplistic KR. As discussed in a previous section currently they cannot capture knowledge that concerns a group of objects, though this may be resolved by the addition of existential and universal quantifiars. Even so, it is doubtful that using this KR *L2Plan* could evolve policies that include recursive concepts. In experiments on the Blocksworld domain, for instance, efficient and effective policies have been evolved but only by adding similar support predicates to those used by Khardon (1999) – the concept of a well-placed block is added to the domain description.

What *L2Plan* currently lacks in KR expressiveness, however, it compensates for by optimising rule order in policies. An iterative rule learning strategy is highly dependent on the training data, which is often biased towards a few actions that occur frequently in plan solving. Since criteria for defining a 'best' rule often concern the number of training examples covered, it is therefore quite likely that the first rules added to any policy dictate the most frequent action found in examples. However, the most frequent actions need not, indeed should not, always be performed first if the aim is an efficient solution. Several crossover and mutation operators in *L2Plan* essentially optimise this aspect of the policy.

This is early-stage work on utilising EC for generalised policy induction and our experiments suggest several avenues for investigation. As indicated the KR is a major theme, and exploring how far we can push a comprehensible though simplistic language, i.e. which domains and which specific features of these domains require a more expressive language, will be highly informative. Description logics and taxonomic syntax are certainly more expressive (at some cost to comprehensibility), and well-worth investigating. It is interesting to note though, that Fern, Yoon, & Givan (2006) cite as a possible reason for their weak policies for the Logistics and Freecell domains a limitation in their KR.

Not explored in this work is *L2Plan*'s potential for also optimising individual rules within a policy. (Khardon 1999), (Martin & Geffner 2004) and (Fern, Yoon, & Givan 2006) all impose limits on the size of rules that may be constructed (as otherwise the search would be prohibitive), thereby restricting a search in the solution space of rules to prespecified regions. One crossover and mutation operation on *L2Plan* rules enables their size to vary, thereby allowing a search in a much wider solution space.

A future improvement is expected from the implementation of typing. The current untyped system means that at least some rules in some policies will be invalid (since predicates can be created that contain variables of the wrong type), presenting lost opportunities for acting on training examples and learning from the evaluation. Typing is therefore expected to reduce the number of iterations necessary to evolve good policies, and/or to present increased opportunities for learning better ones.

Furthermore, analysis of some experiment results also suggest that the current learning process is too highly selective. For instance, only the very best individuals are inserted into the following generation, restricting exploration perhaps too soon in other regions of the search space. This is suggested by the early convergence, and therefore termination of the learning process, to policies that do not perform particularly well on test problems. If the system were allowed to explore a larger area for longer, then it may be possible to evolve better policies.

With regards to improving system efficiency an area of investigation will be the impact of training examples on the quality of the induced policies. A significant computational expense is spent in the production of optimal plans from which to generate training examples. One approach, naturally, is the use of non-optimal planners to generate solutions from which to extract examples. The impact of suboptimal examples on induced policies will therefore also be explored, as empirical studies suggest that a noisy training environment is not necessarily detrimental to the learning process (Ramsey, Schultz, & Grefenstette 1990).

References

Aler, R.; Borrajo, D.; and Isasi, P. 2002. Using genetic programming to learn and improve control knowledge. *Artificial Intelligence* 141:29–56.

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116:123–191.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Fern, A.; Yoon, S.; and Givan, R. 2006. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *Journal of Artificial Intelligence Research* 25:75–118.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:263–302.

Khardon, R. 1999. Learning action strategies for planning domains. *Artificial Intelligence* 113:125–148.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Bradford Book, The MIT Press.

Leckie, C., and Zukerman, I. 1998. Inductive learning of search control rules for planning. *Artificial Intelligence* 101:63–98.

Martin, M., and Geffner, H. 2004. Learning generalized policies from planning examples using concept languages. *Applied Intelligence* 20:9–19.

Miller, B. L., and Goldberg, D. E. 1995. Genetic algorithms, tournament selection, and the effects of noise. Technical Report 95006, Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana, IL.

Pednault, E. 1987. *Toward a Mathematical Theory of Plan Synthesis*. Phd, Stanford University, USA.

Ramsey, C. L.; Schultz, A. C.; and Grefenstette, J. J. 1990. Simulation-assisted learning by competition: Effects of noise differences between training model and target environment. In *Proc. 7th International Conference on Machine Learning*, 211–215.

Spector, L. 1994. Genetic programming and AI planning systems. In *Proc. 12th National Conference on Artificial Intelligence (AAAI-94)*, 1329–1334.

Tesauro, G., and Galperin, G. 1996. On-line policy improvement using Monte-Carlo search. In *Advances in Neural Information Processing 9*.

Assimilating Planning Domain Knowledge from Other Agents

Tim Grant

Netherlands Defence Academy
P.O. Box 90.002
4800 PA Breda, Netherlands
tj.grant@nlda.nl / tgrant@cs.up.ac.za

Abstract

Mainstream research in planning assumes that input information is complete and correct. There are branches of research into plan generation with incomplete planning problems and with incomplete domain models. Approaches include gaining knowledge aimed at making the input information complete or building robust planners that can generate plans despite the incompleteness of the input. This paper addresses planning with complete and correct input information, but where the domain models are distributed over multiple agents. The emphasis is on domain model acquisition, i.e. the first approach. The research reported here adopts the view that the agents must share knowledge if planning is to succeed. This implies that a recipient must be able to assimilate the shared knowledge with its own. An algorithm for inducing domain models from example domain states is presented. The paper shows how the algorithm can be applied to knowledge assimilation and discusses the choice of representation for knowledge sharing. The algorithm has been implemented and applied successfully to eight domains. For knowledge assimilation it has been applied to date just to the blocks world.

Introduction

The plan generation process takes as its input a planning problem consisting of initial and goal states and a domain model typically consisting of planning operators. Its output is a sequence of actions – a plan - that will, on execution, transform the initial state to the goal state.

To locate the research reported here, we place the planning process into its wider context. In Figure 1 the Planning process is central. Its output – a plan – is ingested by the Controlling process. In executing the plan, the Controlling process issues commands to the Process Under Control (PUC), and receives sensory information back.

The Planning process itself has inputs: the domain model and the initial and goal states. The usual assumption is that these inputs come directly from the Controlling process. However, we take the view that each input is developed by an intervening process: initial states result from State Estimation[1], goal states from Goal Setting, and domain models from Modelling. It is these three processes that receive feedback from the Controlling process in the form of the observed sensory information. State Estimation uses the feedback to identify the PUC's current state. Goal Setting determines whether the current goal state has been achieved, can be maintained, or must be replaced by another goal state. Modelling assesses whether the domain model remains a complete and correct description. If not, it uses the feedback to modify or extend the domain model. This paper centres on the Modelling process.

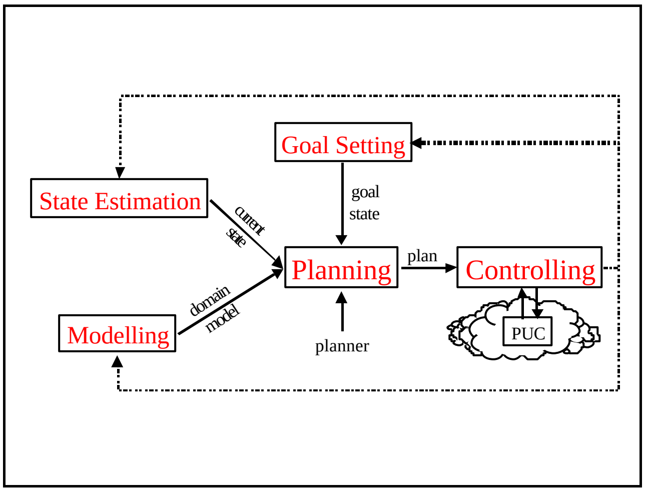


Figure 1. Planning in context.

Mainstream research in planning assumes that the input information is complete and correct[2]. In practical applications, however, information about the domain model, the planning problem, or both may be incomplete and/or incorrect. In the literature there are two approaches to planning with incomplete and/or incorrect input information (Garland & Lesh, 2002):

- Gain better information, either during plan generation or during plan execution. This may be done by using sensors embedded in the PUC to acquire information, by consulting an oracle (e.g. an expert), or by trial-and-error learning from performing experiments in the domain. The acquired information may be used in state estimation, in goal setting, and/or in modelling.
- Build robust planners that can generate plans that succeed regardless of the incompleteness and/or

[1] The term is borrowed from the process control literature.

[2] By convention, the goal state is usually a formula describing a set of (goal) states.

incorrectness of the input information. *Conformant planning* (Goldman & Boddy, 1996) is planning with incomplete knowledge about the initial state and/or the effects of actions. *Model-lite planning* (Kambhampati, 2007) is planning with an incomplete or evolving domain model. *Erroneous planning* (Grant, 2001) has the more limited aim of characterizing the types of erroneous plans generated if the planner is not robust (the error *phenotypes*), based on concepts drawn from the literature on human error, and trying to understand the causes for the observed errors (the error *genotypes*). Knowledge of the error phenotypes and genotypes could then be used for *plan repair* (Krogt, 2005).

By contrast, this paper is concerned about planning with complete and correct input information, but where that information is distributed across multiple agents. In particular, it is concerned with distributed domain models. While the domain model is complete for some set of agents, each individual agent's domain model is (initially) incomplete.

This paper adopts the view that, where knowledge about the planning domain is distributed over multiple agents, the agents must share that knowledge if planning is to succeed. To do so, they must be interoperable. The source of the knowledge and its recipient must adopt a common knowledge representation, as well as coordinating their knowledge-sharing actions. Moreover, the recipient must be capable of assimilating the knowledge gained (Lefkowitz & Lesser, 1988) into other knowledge it may already have. Assimilation of another agent's domain model is an extension of the Modelling process. This paper focuses on the knowledge assimilation capability and choosing a suitable representation for knowledge sharing.

The subject matter in this paper touches on several theoretical areas. Firstly, it is based on the application of machine learning to planning, because knowledge assimilation is a learning process. More specifically, it is concerned with applying machine learning techniques to the acquisition of planning operators. Secondly, because the recipient's domain model is evolving, it touches on model-lite and erroneous planning. Thirdly, it is based on communication theory and, in particular, on information or knowledge sharing concepts drawn from management and organization theory.

The paper is divided into seven chapters. Chapter 2 describes the author's algorithm for modelling planning domains by acquiring planning operators from example domain states. Chapter 3 introduces knowledge sharing based on the Shannon (1948) model of communication. Chapter 4 describes the assimilation of planning domain knowledge, and chooses a suitable representation for sharing that knowledge between agents. Chapter 5 describes two simple worked examples. Chapter 6 surveys related research. Finally, Chapter 7 draws conclusions, identifying the key contributions of this paper, its limitations, and where further research is needed.

Modelling Planning Domains

The author's algorithm for modelling planning domains by acquiring planning operators from example domain states is known as Planning Operator Induction (POI) (Grant, 1996). As the name indicates, POI employs inductive learning from examples. More specifically, it embeds Mitchell's (1982) *version space and candidate elimination* algorithm, taking selected domain states as input examples and inducing STRIPS-style planning operators.

The POI algorithm has been implemented and applied successfully to eight domains (Grant, 1996), including the blocks world, the dining philosophers problem, and a model of a real-world spacecraft payload based on a chemical laboratory instrument. For knowledge assimilation it has been applied to date just to the blocks world.

The ontology employed in POI separates the domain representation into static and dynamic parts. The static part of the POI ontology represents invariant domain entities in terms of object-classes and -instances, of inter-object relationships, and of inter-relationship constraints. By convention, relationships and constraints are binary[3]. For example, the blocks world consists of Hand, Block, and Table object-classes. The `holding` relationship links an instance of the Hand (object-) class to an instance of the Block class, and the `onTable` relationship links a Block instance to a Table instance. The `holding` and `onTable` relationships are constrained in that no Block instance may be both held and on the table simultaneously. Such constraints are known in the planning literature as *domain axioms* or *invariants* and in the database literature as *cardinality* and *exclusion constraints* (Nijssen & Halpin, 1989). The static part of the ontology (less the exclusion constraints) may be depicted using Chen's (1976) Entity-Relationship Diagramming (ERD) notation[4].

The dynamic part of the POI ontology represents domain entity behaviour in terms of states, transitions, and planning operators. Planning operators are reformulations of classes of domain transitions. Instantiated relationships synchronise the states of objects. For example, the `holding hand1 block2` relationship synchronises the states of the objects `hand1` and `block2`: `hand1` must be holding `block2` and `block2` must be held by `hand1` simultaneously. If `hand1` ceases to be holding `block2`, then `block2` must simultaneously cease being held by `hand1`. Transitions combine synchronised changes in relationships. For example, the cessation of the

[3] Higher arity relationships and constraints can be reduced to binary relationships and constraints by changing how the object - and relationship -classes, respectively, are modelled. Details are given in Grant (1996).

[4] The ERD notation is limited to depicting constraints between two instances of the same relationship, i.e. cardinality constraints. It cannot depict constraints between instances of two different relationships, i.e. exclusion constraints. The POI ontology and algorithm is not so limited.

`holding hand1 block2` relationship may be combined with the advent of the `onTable block2 table1` relationship. In terms of Allen's (1983) temporal logic, we would say that the relationships *meet*. Note that they meet because the domain constraint from the previous paragraph forbids them from overlapping. Grant (1995) says that the transition *pivots* around the (instantiated) binary domain constraint.

The POI ontology is similar to McCluskey and Porteus' (1997) object-centred representation for the specification of planning domains. The key differences are that, in POI, the relationships and constraints are strictly binary. Moreover, the constraints hold only between relationships. In addition, objects, relationships, constraints, states, and transitions all have classes, i.e. sorts in McCluskey and Porteus' terminology.

The POI algorithm has two parts:

- *Part 1: Acquisition*. The purpose of the first part of POI is to acquire a static, object-oriented model of the domain from example domain states. POI does not require that the example domain states form a valid sequence, plan-segment, or plan, unlike other algorithms for acquiring planning operators. However, the examples may have to be carefully chosen. Part 1 subdivides into three steps:
  - *Step 1.1*: Acquire domain state description(s).
  - *Step 1.2*: Recognise the objects and relationships in the state description(s).
  - *Step 1.3*: Compile cardinality and exclusion constraints from the objects and relationships. The constraints can be generated exhaustively by constructing all possible pairs between relationships that share an object. For example, pairing the relationship `holding ?Hand1 ?Block1` with `holding ?Hand1 ?Block2` expresses the domain constraint that a hand cannot hold two (or more) blocks simultaneously[5]. By default, a constraint is assumed to hold if no counterexample can be found among the acquired domain state descriptions. Thus, if an agent observed a domain state in which two hands were indeed holding the same block then this constraint would no longer hold.
- *Part 2: Induction*. The purpose of the second part of POI is to induce a dynamic model of domain behaviour from the static, object-oriented domain model. Domain behaviour is modelled using a state-transition network from which planning operators can be extracted. Part 2 sub-divides into six steps:
  - *Step 2.1*: Generate the description language for the domain. The description language is the set of all relationships between object-instances that satisfy the cardinality and exclusion constraints.
  - *Step 2.2*: Construct the version space for the description language using the cardinality and exclusion constraints to eliminate invalid candidate nodes. The version space is a partial lattice of valid nodes, with each node being described in terms of relationships between the domain object-instances.
  - *Step 2.3*: Extract the domain states from the version space. The domain states are the lattice nodes in the maximally specific boundary of the version space.
  - *Step 2.4*: Using the Single Actor / Single State-Change (SA/SSC) meta-heuristic, determine the domain transitions between the domain states. The SA/SSC heuristic is that a single object (the actor) initiates the transition, undergoing a change in just one of its relationships. The actor is at the root of a causal hierarchy of state-changes in the other participating objects. For example, in the blocks world when a robot hand picks up a block from the table, the hand is the actor, making true its `holding` relationship with the block being picked up. The hand's action causes the block both to act on itself so that it is no longer `clear` and to act on the table, breaking the `onTable` relationship.
  - *Step 2.5*: Generalise the domain transitions as transition-classes.
  - *Step 2.6*: Reformat the transition-classes as planning operators.

Depending on how POI is to be used, Part 1 may be optional. If an agent observes an existing domain and uses POI to gain knowledge about how to plan actions in that domain, then Part 1 is essential. By contrast, if an ERD or equivalent static model of a domain (which may not yet exist) is available, then modelling can proceed directly to Part 2. In knowledge assimilation, one agent (the *source*) performs Part 1 and another (the *recipient*) performs Part 2.

Knowledge Sharing

Information and knowledge sharing has been extensively studied in management and organization theory. For simplicity, we will take the terms "information" and "knowledge" as being interchangeable, *pace* Ackoff (1989). Information sharing is a dyadic exchange of information between a source and a recipient (adapted from Szulanski (1996), p.28). Sharing involves the dual problem of "searching for (looking for and identifying) and transferring (moving and incorporating) knowledge across organizational subunits" (Hansen, 1999, p.83). For the purposes of this paper, we will take knowledge sharing as meaning knowledge transfer. Searching for or discovery of other agents that have suitable complementary knowledge about a domain is an area for future research.

Shannon's (1948) model of communication is useful for thinking about knowledge sharing. In the Shannon model, the source and recipient each operate within their own organizational contexts. Information transfer begins when the source generates a message. The message is encoded into a form (a signal) in which it is transmitted by means of a communications medium, such as electromagnetic waves, telephone cables, optical fibres, or a transportable electronic storage medium. Random noise and systematic distortion may be added during

[5] By convention, the same instance of the Block object-class cannot be matched to the two different variables `?Block1` and `?Block2`.

transmission. The recipient decodes the signal and assimilates the decoded message into its own store of knowledge.

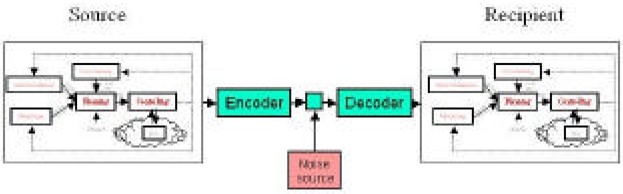


Figure 2. Linking source and recipient agents using Shannon (1948) model.

For the purposes of this paper, we assume that the source and recipient are agents with an internal structure as shown in Figure 1. In general the agents should be able to exchange the outputs of their respective Planning, Controlling, State Estimation, Goal Setting, and Modelling processes, given suitable encoders and decoders (Figure 2). We concentrate here on the Modelling process, how the source's knowledge should be encoded, and what decoder the recipient needs to assimilate that knowledge. We neglect the issue of noise and distortion in this paper.

Assimilating Planning Domain Knowledge

Lefkowitz and Lesser (1988) discuss knowledge assimilation in the context of acquiring domain knowledge from human experts. Their implemented system, $K^{n}_{A}c$, was developed to assist experts in the construction of knowledge bases using a frame-like representation. Assimilated knowledge represented domain objects, relationships, and events. The main contribution of their research was in developing several generic techniques for matching sets of entities and collections of constraints. Research questions included:

- How does the expert's domain description correlate with the description contained in the knowledge base?
- How should the knowledge base be modified based on the expert's new information?
- What should be done when the expert's description differs from the existing one?

Despite the contextual differences, there are strong parallels between Lefkowitz and Lesser's (1988) work and assimilating planning domain knowledge. Assimilation of domain knowledge should be integrated with plan generation and execution. It should permit a variety of ways of learning, including *learning-by-seeing* (i.e. by observing the domain and inferring what actions are possible), *learning-by-being-told* (e.g. by domain experts or other agents), and *learning-by-doing* (i.e. by generating and executing plans). When knowledge is distributed over multiple agents, then individual agents may need to combine different ways of learning. In particular, an agent may well need to combine knowledge it gained from its own observations of a domain with information it has gained by being told by another agent. Like Lefkowitz and Lesser, learning concerns domain objects, relationships, constraints, and events. Analogues of Lefkowitz and Lesser's research questions apply; here we are concerned with the planning analogue of their second question.

Considering the POI algorithm from the viewpoint of encoding and decoding, we see that there are three forms in which knowledge relating to the domain model could be exchanged:

- *As cases*. The source agent could transmit the domain states it has observed, i.e. the input information to Part 1 of the POI algorithm. The source agent would not have to process its observations before transferring them to the recipient. The recipient agent would then have to add the source's domain states to its own database of domain states, and perform Parts 1 and 2 of the POI algorithm to obtain a set of planning operators. Exchanging knowledge in this form is likely to be verbose for real-world domains, possibly with duplicated observations. More importantly, it would limit knowledge assimilation to learning-by-seeing. The only thing that knowledge sharing achieves is that the recipient can "see" both what it can itself observe and what the source has observed.
- *As static domain models*. The source agent could transmit its static domain model, i.e. the information as output by Part 1 and as input to Part 2. The source agent would have had to perform Part 1 before transmitting its static domain model to the recipient. The recipient agent would then have to add the source's objects, relationships, and constraints to its own database of objects, relationships, and constraints. Where source and recipient agents disagree on whether a constraint holds, then the constraint is assumed not to hold (because one of the agents will have seen a counterexample). The recipient retains its own list of object-instances and does not assimilate the source's object-instances list, because the recipient may not be able to execute plans on objects that it cannot see. Then the recipient would perform Part 2 of the POI algorithm to obtain a set of planning operators. Exchanging knowledge in this form is likely to be concise. Moreover, it would allow learning-by-seeing, learning-by-being-told, and their combination.
- *As planning operators*. The source agent could transmit its dynamic domain model, i.e. the information as output by Part 2. The recipient agent would then simply have to add the planning operators obtained from the source to its own planning operators. Exchanging knowledge in this form is still more concise, but assumes that (1) the source and the recipient agents' observations are sufficiently rich for both of them to be able to induce a set of planning operators, and that (2) their sets of planning operators are complementary. There is no way for additional planning operators to be induced by synergy.

In this research, the encoding-decoding schema has been determined by the researcher. Ideally, the source and recipient agents should themselves be able to negotiate a suitable encoding-decoding schema, depending on considerations such as privacy, security, and communications bandwidth. Further research is needed to provide agents with such a capability.

Worked Examples

Two worked examples should make the key issues clear. The first example is the one-block world and the second is taken from the three-blocks world. Because the one-block world is simple, the first example is described in more detail. The second example illustrates the need to select example states carefully if the agents are to induce a full set of planning operators.

Suppose two agents each observe a different state of a one-block world (Slaney & Thiebaux, 2001), as represented by Nilsson (1980)[6]. There are two possible states[7]: [[holding hand1 block1] [onTable block1 nil] [onTable nil table1]] and [[holding hand1 nil] [holding nil block1] [onTable block1 table1]]. Let us suppose that Agent1 is given the first state description and Agent2 the second.

The following table depicts the static domain model that would result from their performing Part 1 separately, i.e. without knowledge sharing and assimilation:

	Agent1's model	**Agent2's model**
Object-classes	Hand, Block, Table	Hand, Block, Table
Object-instances	hand1, block1, table1	hand1, block1, table1
Relations	holding ?Hand ?Block onTable ?Block nil onTable nil ?Table	holding ?Hand nil holding nil ?Block onTable ?Block ?Table
Constraints	IF holding ?Hand1 ?Block1 AND holding ?Hand1 ?Block2 THEN INVALID -- hand cannot hold multiple blocks IF holding ?Hand1 ?Block1 AND holding ?Hand2 ?Block1 THEN INVALID -- block cannot be held by multiple hands IF holding ?Hand1 ?Block1 AND onTable ?Block1 nil THEN INVALID -- block cannot be held and not on a table NOTE: This constraint does not hold because the observed state is a counterexample. IF onTable ?Block1 nil AND onTable ?Block2 nil THEN INVALID -- multiple blocks cannot be off the table IF onTable nil ?Table1 AND onTable nil ?Table2 THEN INVALID -- multiple tables cannot be clear at same time	IF holding ?Hand1 nil AND holding ?Hand2 nil THEN INVALID -- multiple hands cannot be empty at same time IF holding nil ?Block1 AND holding nil ?Block2 THEN INVALID -- multiple blocks cannot be not held IF holding nil ?Block1 AND onTable ?Block1 ?Table1 THEN INVALID -- block cannot be not held and on a table NOTE: This constraint does not hold because the observed state is a counterexample. IF onTable ?Block1 ?Table1 AND onTable ?Block1 ?Table2 THEN INVALID -- block cannot be on multiple tables IF onTable ?Block1 ?Table1 AND onTable ?Block2 ?Table1 THEN INVALID -- table cannot hold multiple blocks

Neither of the agents would be able to induce any planning operators, because POI Part 2 would simply result in the induction of a single state, namely the state each agent had observed originally. There needs to be a minimum of two states for the SA/SSC heuristic to find any transitions.

Now suppose that the agents share their domain knowledge. Since neither of them can induce planning operators separately, exchanging data in the form of planning operators is not feasible. However, they can exchange knowledge in the form either of cases or of their static domain models. For the one-block world it is simpler for the agents to exchange cases, but this does not apply to complex, real-world examples.

Sharing their domain models enables the agents to create synergistic knowledge. Firstly, Agent1 learns from Agent2 that blocks can be on tables, and Agent2 learns from Agent1 that hands can hold blocks. Secondly, additional constraints can be identified, as shown in the following table:

	Synergistic knowledge
Relations	holding ?Hand ?Block holding ?Hand nil holding nil ?Block onTable ?Block ?Table onTable ?Block nil onTable nil ?Table
Constraints	IF holding ?Hand1 nil AND holding ?Hand1 ?Block1 THEN INVALID -- hand cannot be both empty and holding a block IF holding nil ?Block1 AND holding ?Hand1 ?Block1 THEN INVALID -- hand cannot be both held by a hand and not held IF holding ?Hand1 ?Block1 AND onTable ?Block1 ?Table1 THEN INVALID -- block cannot be both held and on a table IF onTable nil ?Table1

[6] Distinguishing three object-classes (Hand, Block, Table) and yielding four operators (pickup, putdown, stack, unstack). See Grant et al, 1994.

[7] A third state would be observed in an orbiting spacecraft: [[holding hand1 nil] [holding nil block1] [onTable block1 nil] [onTable nil table1]]. During development of the POI algorithm the three states were indeed induced, resulting in the induction of a set of six operators (pickup, putdown, floatoff, floaton, letgo, capture). The author observed that he had failed to represent the action of gravity. To do so while retaining the Nilsson (1980) domain representation requires a triple constraint, stating in effect that a block must be either held by a hand or supported by a table or by another block. This can be solved by extending the POI ontology, either by allowing constraints of arity higher than two or by introducing an inheritance hierarchy of object-classes. The author adopted the latter solution, because this has the synergistic consequence of reducing the complexity of the version space, leading to savings in induction time and memory requirements (Grant, 1996).

	AND onTable ?Block1 ?Table1 THEN INVALID -- table cannot be supporting both a block and nothing IF onTable ?Block1 nil AND onTable ?Block1 ?Table1 THEN INVALID -- block cannot be both off and supported by a table

The synergistic knowledge, together with the additional constraints, enables the agents to induce the `pickup` and `putdown` planning operators. They do not have enough knowledge to induce the `stack` and `unstack` operators because stacks of blocks and the `on` relationship between blocks does not exist in the one-block world.

The three-blocks world has 22 states, falling into five state-classes (Grant et al, 1994). Experiments with the implemented POI algorithm, adapted for knowledge assimilation, showed that it is not necessary for the agents to observe all 22 states (Grant, 1996). Just two, judiciously-chosen, example states sufficed[8]. In one state the hand must be empty, and in the other it must be holding a block. One state must show a stack of at least two blocks, and one stack must show two or more blocks on the table. Inspection shows that there are four pairings of the five state-classes that can meet these requirements. Two can be rejected on the grounds that they are *adjacent*, i.e. that they are separated by the application of just one operator. Successful knowledge assimilation has been demonstrated for the remaining two state-pairs: for all three blocks on the table paired with the state in which one block is held and the other two are stacked, and for a stack of three blocks paired with the state in which one block is held and the other two are on the table. Moreover, the induced set of planning operators can be used to generate and successfully execute a plan that passes through at least one novel state, i.e. a state that the agents had not previously observed.

It is not known whether two (judiciously-chosen) example states suffice in all domains for the induction of a full set of planning operators. Hand simulations and experiments have only been done for the (one-hand, one-table, and) one- and three-blocks worlds. More research is needed, e.g. by applying knowledge assimilation using POI to the International Planning Competition benchmark domains and to real-world domains where planning knowledge is distributed geographically or organizationally.

Related Work

In 2003, Zimmerman and Kambhampati surveyed the research on applying machine learning to planning. They identified three opportunities for learning: before planning, during planning, and during execution. Learning techniques applied fell into two groups: inductive versus deductive (or analytical) learning. Inductive techniques used included decision tree learning, neural network, inductive logic programming, and reinforcement learning. They observed that early research emphasised learning search control heuristics to speed up planning. This has fallen out of favour as faster planners have become available. There is now a trend towards learning or refining sets of planning operators to enable a planner to become effective with an incomplete domain model or in the presence of uncertainty. "Programming by demonstration" can be applied so that the user of an interactive planner could create plans for example problems that the learning system would then parse to learn aspects peculiar to the user.

In terms of Zimmerman and Kambhampati's (2003) survey, this paper applies Mitchell's (1982) inductive version space and candidate elimination algorithm to planning. The POI algorithm could be used before planning, during planning, or during execution. It centres on the learning of domain models in the form of planning operators. It exhibits an element of "programming by demonstration" in that the user shows POI example domain states, rather than example plans or execution traces.

In his 2006 lectures on learning and planning at the Machine Learning Summer School, Kambhampati distinguished three applications of learning to planning: learning search control rules and heuristics, learning domain models, and learning strategies. Research in learning domain models could be classified along three dimensions: the availability of information about intermediate states, the availability of partial action models, and interactive learning in the presence of humans. POI does not need information about intermediate states nor partial action models, and it does not require the presence of humans. By comparison, other operator learning algorithms require as input:.

- Background domain knowledge: Porter & Kibler (1986), Shen (1994), Levine & DeJong (2006).
- Partial domain model (i.e. operator refinement, rather than *ab initio* operator learning): Gil (1992), DesJardins (1994), McCluskey et al (2002).
- Example plans or traces: Oates and Cohen (1996), Wang (1996), Yong et al (2005).
- Input from human experts: McCluskey et al (2002). POI can accept a static domain model from a human expert (e.g. for a domain that does not yet exist) instead of observing domain states, but this is not applicable to assimilating domain knowledge distributed over multiple agents.

POI is closest to Mukherji and Schubert (2005) in that it takes state descriptions as input and discovers planning invariants. The differences are that POI also discovers objects and relationships and uses the information it has discovered to induce planning operators. Like McCluskey and his collaborators (McCluskey & Porteus, 1997; McCluskey et al, 2002), POI models domains in terms of object-classes (sorts, in McCluskey's terminology), relationships, and constraints.

[8] Introspection suggests that there may be a single state in the two-hands, four-block world that could provide all the information needed to induce all four operators, but then the knowledge could not be distributed over multiple agents.

Conclusions

This paper has addressed the topic of planning with a domain model that is complete and correct but distributed across multiple agents. The paper takes the view that the agents must share their knowledge if planning is to succeed. The Planning Operator Induction (POI) algorithm (Grant, 1996) has been introduced as a means of acquiring planning operators from carefully-chosen examples of domain states. Unlike other algorithms for acquiring planning operators (Porter & Kibler, 1986) (Gil, 1992) (Shen, 1994) (DesJardins, 1994) (Wang, 1996) (Oates & Cohen, 1996) (McCluskey et al, 2002) (Yang et al, 2005) (Mukherji & Schubert, 2005) (Levine & DeJong, 2006), the example domain states do not need to form a valid sequence, plan-segment, or plan, nor do preceding or succeeding transitions have to be given. When agents share their partial knowledge of the domain model, the two parts of the POI algorithm can be divided between the source and recipient in the knowledge-sharing process. The agents exchange the static, object-oriented domain model resulting from Part 1 of the POI algorithm. This enables the recipient to identify synergies between the shared knowledge and knowledge it already has and to perform the induction, i.e. Part 2 of the algorithm.

This paper makes several contributions. Its primary contribution is in showing how planning domain knowledge that is distributed across multiple agents may be assimilated by sharing partial domain models. Secondary contributions include:

- The POI domain-modelling algorithm is presented that acquires planning operators from example domain states. The example domain states do not need to form a valid sequence, plan-segment, or plan, nor do preceding or succeeding transitions have to be given.
- The ontology used in the POI algorithm extends McCluskey and Porteus' (1997) object-centred representation. Relationships and constraints are strictly binary. Constraints are between pairs of relationships, rather than domain-level axioms. Hence, both relationships and constraints are associated with (classes of) domain objects.

A key limitation of the research reported here is that, while knowledge assimilation using the POI algorithm has been implemented, it has only been tested for the (one-hand, one-table, and) one- and three-blocks worlds. Future research should include:

- Applying POI-based knowledge assimilation to a wider variety of planning domains, e.g. International Planning Competition benchmark domains. One research question to be addressed is whether two (judiciously-chosen) example states suffice in all domains for the induction of a full set of planning operators.
- Elucidating the conceptual links between the POI algorithm and plan generation using planning graphs.
- Applying the POI algorithm to sense-making, i.e. the modelling of novel situations (Weick, 1995). An approach has been outlined in Grant (2005).
- Extending the POI ontology to model inheritance and aggregation relationships, with the eventual aim of using the Unified Modeling Language (UML) as a representation for the static, object-oriented and dynamic, behavioural domain models in the POI algorithm.
- Developing an integrated planning environment that incorporates domain modelling, plan generation, plan execution, state estimation, and goal setting to act on real and simulated domains.
- Extending agent capability to (1) negotiating mutually-acceptable encoding-decoding schemes, and (2) discover agents that have complementary knowledge.
- Investigating the application of knowledge assimilation using POI to real-world domains where planning knowledge is distributed geographically or organizationally. Example domains include air traffic control and military Command & Control (Grant, 2006).

References

Ackoff, R. 1989. *From Data to Wisdom*. Journal of Applied Systems Analysis, 16, 3-9.

Allen, J.F. 1983. *Maintaining Knowledge about Temporal Intervals*. Communications of the ACM, 26, 11, 832-843.

Chen, P.P-S. 1976. *The Entity-Relationship Model: Towards a unified view of data*. ACM Transactions on Database Systems, 1, 9-36.

DesJardins, M. 1994. *Knowledge Development Methods for Planning Systems*. Proceedings, AAAI-94 Fall Symposium series, Planning and Learning: On to real applications. New Orleans, LA, USA.

Fikes, R., & Nilsson, N.J. 1971. *STRIPS: A new approach to the application of theorem proving to problem solving*. Artificial Intelligence Journal, 2, 189-208.

Garland, A., & Lesh, N. 2002. *Plan Evaluation with Incomplete Action Descriptions*. TR2002-05, Mitsubishi Electric Research Laboratories, Cambridge, Massachusetts, USA.

Gil, Y. 1992. *Acquiring Domain Knowledge for Planning by Experimentation*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.

Goldman, R., & Boddy, M. 1996. *Expressive Planning and Explicit Knowledge*. Proceedings, AIPS-96, 110-117, AAAI Press.

Grant, T.J. 1995. *Generating Plans from a Domain Model*. Proceedings, 14th workshop of the UK Planning and Scheduling Special Interest Group, 22-23 November 1995, University of Essex, Colchester, UK.

Grant, T.J. 1996. *Inductive Learning of Knowledge-Based Planning Operators*. PhD thesis, University of Maastricht, The Netherlands.

Grant, T.J. 2001. *Towards a Taxonomy of Erroneous Planning*. Proceedings, 20th workshop of the UK Planning and Scheduling Special Interest Group, 13-14 December 2001, University of Edinburgh, Scotland.

Grant, T.J. 2005. *Integrating Sensemaking and Response using Planning Operator Induction*. In Van de Walle, B.

& Carlé, B. (eds.), Proceedings, 2nd International Conference on Information Systems for Crisis Response and Management (ISCRAM), Royal Flemish Academy of Science and the Arts, Brussels, Belgium, 18-20 April 2005. SCK.CEN and University of Tilburg, 89-96.

Grant, T.J. 2006. *Measuring the Potential Benefits of NCW: 9/11 as case study*. In Proceedings, 11th International Command & Control Research & Technology Symposium (ICCRTS06), Cambridge, UK, paper I-103.

Grant, T.J., Herik, H.J. van den, & Hudson, P.T.W. 1994. *Which Blocks World is* the *Blocks World?* Proceedings, 13th workshop of the UK Planning and Scheduling Special Interest Group, University of Strathclyde, Glasgow, Scotland.

Hansen, M. T. 1999. *The Search-Transfer Problem: The role of weak ties in sharing knowledge across organization subunits*. Administrative Science Quarterly, 44 (1), 82-111.

Kambhampati, S. 2006. *Lectures on Learning and Planning*. 2006 Machine Learning Summer School (MLSS'06), Canberra, Australia.

Kambhampati, S. 2007. *Model-lite Planning for the Web Age Masses: The challenges of planning with incomplete and evolving domain models*. Proceedings, American Association for Artificial Intelligence.

Krogt, R. van der. 2005. *Plan Repair in Single-Agent and Multi-Agent Systems*. PhD thesis, TRAIL Thesis-series T2005/18, TRAIL Research School, Netherlands.

Lefkowitz, L. S., and Lesser, V. R. 1988. *Knowledge Acquisition as Knowledge Assimilation*. International Journal of Man-Machine Studies, 29, 215-226.

Levine, G., & DeJong, G. 2006. *Explanation-Based Acquisition of Planning Operators*. Proceedings, ICAPS 2006.

McCluskey, T.L., & Porteus, J.M. 1997. *Engineering and Compiling Planning Domain Models to Promote Validity and Efficiency*. Artificial Intelligence Journal, 95, 1-65.

McCluskey, T.L., Richardson, N.E., & Simpson, R.M. 2002. *An Interactive Method for Inducing Operator Descriptions*. Proceedings, ICAPS 2002.

Mitchell, T.M. 1982. *Generalization as Search*. Artificial Intelligence Journal, 18, 203-226.

Mukherji, P., & Schubert, L.K. 2005. *Discovering Planning Invariants as Anomalies in State Descriptions*. Proceedings, ICAPS 2005.

Nijssen, G.M., & Halpin, T.A. 1989. *Conceptual Schema and Relational Database Design: A fact-oriented approach*. Prentice-Hall Pty Ltd, Sydney, Australia.

Nilsson, N.J. 1980. *Principles of Artificial Intelligence*. Tioga Publishing Company, Palo Alto, California, USA.

Oates, T., & Cohen, P.R. 1996. *Searching for Planning Operators with Context-Dependent and Probabilistic Effects*. Proceedings, AAAI, 865-868.

Porter, B., & Kibler, D. 1986. *Experimental Goal Regression: A method for learning problem-solving heuristics*. Machine Learning, 1, 249-284.

Shannon, C.E. 1948. *A Mathematical Theory of Communication*. Bell System Technical Journal, 27, 379-423 (July) & 623-646 (October).

Shen, W.-M. 1994. *Discovery as Autonomous Learning from the Environment*. Machine Learning, 12, 143-156.

Slaney, J., & Thiébaux, S. 2001. *Blocks World Revisited*. Artificial Intelligence Journal, 125, 119-153.

Szulanski, G. 1996. *Exploring Internal Stickiness: Impediments to the transfer of best practice within the firm*. Strategic Management Journal, 17, 27-43.

Wang, X. 1994. *Learning Planning Operators by Observation and Practice*. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA.

Weick, K. 1995. *Sensemaking in Organizations*. Sage, Thousand Oaks, CA, USA. ISBN 0-8039-7178-1.

Yang, Q., Wu, K., & Jiang, Y. 2005. *Learning Action Models from Plan Examples with Incomplete Knowledge*. Proceedings, ICAPS 2005, 241-250.

Zimmerman, T., & Kambhampati, S. 2003. *Learning-Assisted Automated Planning: Looking back, taking stock, going forward*. AI magazine, 73-96 (Summer 2003).

The Dimensions of Driverlog

Peter Gregory and **Alan Lindsay**
University of Strathclyde
Glasgow
UK
{pg|al}@cis.strath.ac.uk

Abstract

The International Planning Competition has provided a means of comparing the performance of planners. It is supposed to be a driving-force for planning technology. As the competition has advanced, more and more complex domains have been introduced. However, the methods for generating the competition instances are typically simplistic. At best, this means that our planners are not tested on the broad range of problem structures that can be expressed in each of the domains. At worst, it means that some search techniques (such as symmetry-breaking and graph-abstraction) are ineffective for the competition instances.

It is our opinion that a competition with interesting instances (those with varied structural properties) would better drive the community to developing techniques that address real-world issues, and not just solving contrived competition test-cases. Towards this end, we present a preliminary problem generator for the Driverlog domain, and introduce several important qualities (or dimensions) of the domain. The performance of three planners on instances generated by our generator are compared with their performance on the competition instances.

Introduction

The International Planning Competitions have been a driving force for the development of planning technology. Each competition in turn has added to the expressivity of the standard language of AI Planning: PDDL. The domains that have been created for each competition have also increased in complexity and structure. For domains tested in the early planning competitions, such as Blocksworld, problem generation was not considered a difficult problem: generate two random configurations of the blocks and use those as the initial and goal states.

Slaney and Thiebaux showed that even for Blocksworld, problem generation is an interesting problem. Using the intuitive technique to generate states will not generate all possible states (Slaney & Thiébaux 2001). If a simple, intuitive problem generation strategy is not satisfactory for a domain such as Blocksworld, it seems highly unlikely that a similar strategy would be satisfactory for a modern, highly-structured domain.

This work addresses two questions. The first is how to generate an interesting benchmark set for a complex structured domain (the Driverlog domain). The second question asks whether or not the competition results accurately reflect the performance of the competing planners across the benchmark problems that have been created.

Ideally, a set of benchmarks should test current planning technology to its limits. More than simply supplying problems that reach outside of the scope of current planners, a benchmark set should highlight the particular structural properties that planners struggle with. This provides focus for future research. Studying the reasons why our planners fail to solve certain types of problems reveals where future improvements might be made.

Benchmarks should, when appropriate, model reality in a useful way. Of course, it is infeasible to expect planners to solve problems on a massive scale. But it is possible to retain structural features of real-world problems. Nobody would write a logistics instance in which a particular package was in more than one location in the initial state, although this would probably be allowed by the domain file. The structural property that objects cannot occupy more than one location is intuitive, but there may be other real-world structural properties that are not as obvious.

The final function that a good benchmark set should provide is a solid foundation for critical analysis of different planners. One criticism of the IPC could be that there are simply not enough instances to know which planner is best and when. Ideally, there should be enough results to prove that some planner is faster, or produces higher quality plans to a statistically significant level.

The Driverlog Problem

A transportation problem involves a set of trucks moving packages from a starting location to a goal destination in an efficient way. The trucks drive along roads that connect the locations together, and a package can be picked up from or dropped off to a truck's current location. The Driverlog domain extends this model by introducing drivers. Drivers have their own path network that connects the locations together, allowing them to walk between locations. Trucks in Driverlog can only move if they are being driven by a driver. This introduces an enabler role moving away from a simple deliverable/transporter model. As well as this, goal locations are often set for the drivers and the trucks, not just the packages.

Transportation domains can cover problems with interesting road structures. However, Driverlog adds interesting challenges, as there can be complicated interaction between the two graphs structures and there are many more factors to consider when deciding how to deliver the packages, including additional goal types and useful driver truck pairings.

The Dimensions of Driverlog

A Driverlog problem comprises the following things: a set of drivers, a set of trucks, a set of packages and a set of locations. All of the drivers, trucks and packages are initially at a location. A subset of the drivers, trucks and packages have a goal location. Locations are connected in two distinct ways: by paths (which drivers can walk along) and by roads (which trucks can drive along).

We propose eight dimensions that we feel could be combined to create interesting and challenging Driverlog problems. The dimensions largely focus on introducing structural features to the graphs, however, we also consider the types of goals and number of the separate objects in the problem. These could greatly affect the difficulty of the problem.

Graph topology There are several options relating to the graph topology, connectivity and planar or non-planar. Planar graphs are graphs that can be drawn on a plane, with no intersecting edges, a property existing in many real road networks. This domain can be used to represent other problems and it is likely that non-planar graphs will also be of interest and increase the problem space. The connectivity of the graph, from sparse to dense can also be set, allowing a whole range of interesting structures to be explored.

Numbers of objects The number of trucks, drivers, packages and locations are the traditional parameters for generating Driverlog problems. This dimension can be used to set the size of the problem and can have some effect on the difficulty.

Types of goals There are only three possible types of goal in Driverlog: the goal location for trucks, drivers or packages. In real world transportation problems, the planner can never consider delivering the packages in isolation; the final destination of the drivers and trucks is also extremely important. Allowing the types of goals to be selected provides control over the emphasis of the problem.

Disconnected drivers There are two separate graphs in the Driverlog domain, the road graph and the path graph. The interesting interactions that can happen between the two graph structures are usually ignored. We want to encourage exploration of these interactions. Disconnected drivers provide problems where drivers must traverse both graphs (walking, or in a truck) to solve the problem.

One-way streets Links can be added between two locations in a single direction. This means that trucks can move from one location to another, but may have to find a different route to return to the original location. Solving problems with directed graph structure forces careful planning of how the trucks are moved around the structure. If the wrong truck is moved down one of the one-way streets, then many wasted moves could be incurred as the truck traverses back through the graph. As well as adding an interesting level of difficulty, we think this dimension is particularly relevant, because of the increasing number of one-way streets in the transport network.

Dead ends Dead ends are locations, or groups of locations that are joined to the main location structure in one direction only. This means that a truck cannot return once it has moved into one of these groups of locations. This forces the planner to carefully decide when to send the truck into one of these groups, as it will be lost for the remainder of the plan. For example, on difficult terrain there can be craters that a robot can manage to move into, but are too steep for the robot to get out again. In this case the planner may want to balance the importance of the scientific gain with the cost of the robot and termination of the mission.

SAT/ UNSAT This dimension allows the possibility of unsolvable problems. Solvable means that there is a sequence of actions moving the state from the initial state to a state that satisfies the goal formula. This option might allow the exploration of more interesting properties in the other dimensions, as sometimes it is impossible to ensure that certain combinations are solvable.

Symmetry in objects Symmetry occurs in the Driverlog problem when different objects or configurations of objects are repeated. For example, three trucks that have the same start and goal conditions are symmetric. Also, the underlying road network may be symmetric. Planners that perform symmetry breaking can exploit symmetry to reduce the amount of necessary search.

The Instance Generators

Four different generators have been written for this work. In the future, these will be reduced to a single generator. But since this is preliminary work, different generators were produced for different important dimensions. These are Planar, Non-Planar, Dead-ends and Disconnected Drivers. The generators explore the different dimensions identified as interesting in the previous section. Three of these dimensions are not explored: Symmetry in objects, types of goals and SAT/UNSAT. The majority of modern planners have been built around the assumption that instances will be satisfiable, and so this dimension may not produce any interesting discussion. In all of the instances, each driver, truck and package has a goal destination (unless otherwise specified). Symmetry in objects cannot be explicitly varied in any of the generators. It is our intention to add the capacity to vary these dimensions in the future. One more restriction is that except in the Disconnected Drivers generator, the path map is identical to the link map. The generators do the following:

Planar

Generates instances with planar maps. The user can vary the number of drivers, trucks, packages and locations. The user is required to supply the probability of two locations being connected. The user specifies if the map is directed or not. All of the generated maps will be connected. In the implementation, if a generated map is not connected, it is simply discarded and a new one generated.

Non-Planar

The Non-Planar generator is similar to the Planar generator except that the user specifies a particular number of links in the road-map and, of course, the resultant road-maps may not be planar.

Dead-ends

To test road maps with dead-ends, the following method is used for generating an instance. A tree is constructed as the road map randomly, connecting location n with a random location, lower than n. There are t trucks and drivers, initially located at location 1. The last t locations are then used as destination locations for the packages. The trucks do not have a destination location specified.

Each package is randomly assigned a destination from those last t locations. Each package is then initially placed at any location on the path between location 1 and its destination. The challenge in this problem is simply to drive a truck to each destination and only load a truck with packages that are supposed to be delivered to that truck's destination. Figure 1 shows one example. In this example, normal fonts represent the initial location of packages, italicised fonts represent their goal locations.

Disconnected Drivers

The Disconnected Driver generator is designed to explore the Disconnected Driver dimension. In order to do this, a map with no paths is created. Each driver is paired with a truck: the goal locations of the truck and driver are the same. Their initial locations are not the same (although each driver has a truck available). The challenge in the instances generated is in swapping the drivers into the truck that shares its goal location.

Experiments

To test the generators, we have used three of the most successful planners of recent times, FF (Hoffmann & Nebel 2001), LPG (Gerevini & Serina 2002) and SGPlan (Chen, Wah, & Hsu 2006). We used FF version 2.3, LPG version 1.2 and SGPlan version 41. All of the tests were performed on a desktop computer with a dual-core Intel Pentium 4 2.60GHz CPU. The tests were limited to using 10 minutes and 300MB of memory. The timings for FF and LPG measure system time + user time. Sadly, there is no simple way of calculating this measure for SGPlan, and so clock time is used. This could mean that SGPlan seems slightly slower than in reality. However, system load was minimal during testing, and any scaling in performance should be a very small constant factor. The quality of plans is measured by number of actions. As FF only produces sequential plans, and LPG by default optimises number of actions, this was thought a fairer measure than makespan.

We generated a huge number of benchmark test cases, and then after some preliminary small-scale tests chose an interesting selection of problems that covered a range of difficulty for all of the planners. We highly recommend this method. Without preliminary tests, it is impossible to know what range of problems may provide difficulties for the planners. It is far too easy to construct a benchmark set composed entirely of either very easy or impossible to solve problems.

We provide detailed results for planar road maps with four drivers, four trucks, nine packages, and number of locations varying between 10 and 30, in steps of five. For each size of map, we generated 50 instances with probability of two nodes being connected of between 0.1 and 0.9 both for directed and undirected graphs. This gives 250 instances for directed and undirected graphs. Planar graphs were selected as they have a similar structure to real-world road networks.

We also used 180 of the generated Dead End instances. These instances have between one and four trucks, they have 9 packages and all have 15 locations.

Results

The results of performing the above experiments can be seen in Figure 2 to Figure 5. These graphs show the planar directed results (both time and quality) for FF vs. LPG, FF vs. SGPlan and LPG vs. SGPlan respectively. The graphs of the timings are log-scaled, whereas the graphs showing quality are linear scaled.

Time vs. Quality

The results shown in Figure 2 to Figure 5 show that there is little to choose between the planners in terms of plan quality. In each comparison, the two compared planners seem to gain wins in what seems about half of the cases. However, of the three planners, LPG is considerably better in terms of time taken than the other planners. This highlights the fact that planners have been built specifically with the task of attaining satisfiability, rather than trying to optimise metrics.

SGPlan Dependence on FF

It was noticed that in many problems that were found difficult by FF, SGPlan also struggled. FF's performance seems to dominate that of SGPlan. This is unusual, as each goal in a Driverlog problem should be reasonably straightforward to solve in isolation. However, it is perhaps due to the fact that some of the goals in Driverlog are strongly dependent on resources that participate in other goals. This could mean that combining the sub-plans becomes difficult for SGPlan.

Dead-end Example

Figure 1 shows an example of the Dead End instances generated. This instance had three trucks. The numbers in Figure 1 represent package locations. The italicised numbers represent the goal locations of the packages. All three planners were incapable of solving this simple problem.

This highlights the fact that the planners do not reason about resource allocation intelligently. If the problem is viewed as a task of assigning packages to trucks, then the problem becomes very simple. It also shows that the planners do not reason about the consequences of irreversible actions.

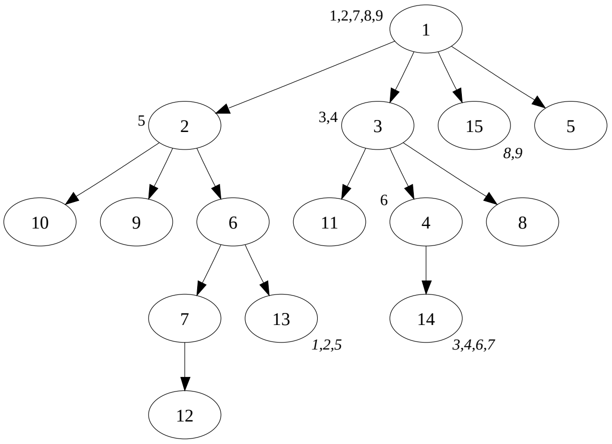


Figure 1: Dead-end instance in which all three planners fail

Instance	1	2	3	4	5	6	7	8	9	10
FF	9	23	13	17	23	14	18	24	32	21
LPG	7	27	13	16	31	14	21	25	32	22
SGPlan	9	24	13	21	24	14	18	30	35	19
Instance	11	12	13	14	15	16	17	18	19	20
FF	26	53	37	39	49	–	–	–	–	–
LPG	25	42	33	78	60	284	143	179	230	176
SGPlan	25	39	36	44	47	–	–	–	–	–

Table 1: Plan Quality for the 2002 IPC Benchmark Instances

Directed vs. Undirected

Figure 2 and Figure 3 show the results of the Planar Directed and Undirected tests respectively. For each of the planners, there was no large difference in the results between the directed and undirected test cases. It was thought that for the same reason the planners deal badly with dead-ends, they may also deal badly with one-way streets. This appears not to be the case, although further experiments may reveal more specific forms of dead-end roads in which the planners struggle.

Competition Comparisons

The planning competition provides a strong motivation in our field and directs the activity of the community. In this study we examined the generator used in the 2002 IPC (Long & Fox 2003): the Driverlog problem generator. We feel that the generator does not provide problems that capture the full potential of what is a structurally rich domain. Therefore it is our opinion that the competition has failed to fully explore how the planners behave in this domain. Our approach focusses on generating problems with several varied structural features and we feel our results provide more understanding of the planners' strengths and weaknesses. We believe that this provides a far stronger base for making comparisons between the planners. In this section we describe the Driverlog generator used in the competitions and discuss the differences between the results of the competition and the results found in this study.

Instance	1	2	3	4	5	6	7	8	9	10
FF	0	0	0	0	0	0	0	0	0	0
LPG	0	0	0	0	0	0	0	0	0	0
SGPlan	0	0	0	0	0	0.1	0	0	0	0
Instance	11	12	13	14	15	16	17	18	19	20
FF	0	0.4	0.2	0.3	0.1	–	–	–	–	–
LPG	0	0.1	0.1	0.2	0.4	85.9	2.8	8.1	52.1	72.1
SGPlan	0	0.2	0.1	0.1	0.1	–	–	–	–	–

Table 2: Execution Time for the 2002 IPC Benchmark Instances

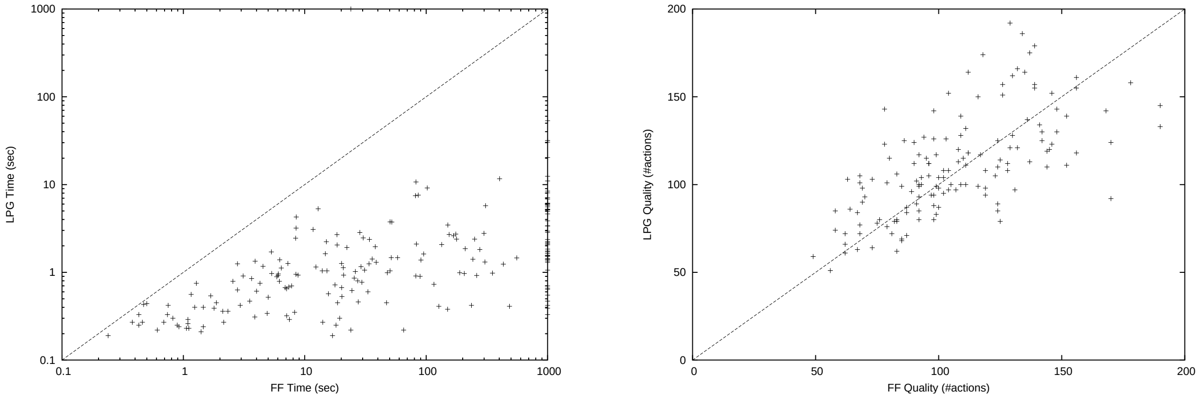
Driverlog is a domain that is rich in structure, however the current competition generator uses a very simple approach to creating the test cases. The parameters to the generator, are the number of trucks, drivers, packages and road locations. The connectivity of the road graph is determined by making (number of road locations $\times$ 4) connections between any two random locations. If the graph is not connected, then additional connections are added between disconnected locations until it is. It is highly likely that this method will produce a very densely connected graph. The same happens for the path graph, except there are (2 $\times$ the number of locations) instead, thus increasing the chances of a sparser path graph. These graphs are both undirected, removing any chances of one way streets or dead-ends and each cover all the road locations, removing the possibility of disconnected drivers. As the graphs are so densely connected it is unlikely that they will be planar and even less likely that they will resemble real-world road networks.

The objects are positioned randomly across the locations and their goal locations (if required) are chosen randomly too. The decision on whether an object has a goal is randomly made, with 95% chance of a package having a goal destination and 70% for both drivers and trucks. This means that no control is given to the types of goal in the problem and no effort is made to position the goals in an interesting way.

We feel that the planning competition should be able to prove that a planner is faster or produces better quality plans to a statistically significant level. Also, that how a planner performs in a particular area of planning should be identifiable. In our approach we generated problems that incorporated several interesting structural features and spanned a whole range of difficulties. This provides a solid base for judging the performance of the planners across the whole domain and additionally provides invaluable insight into how the planner behaves when faced with specific structural features. We believe that the competition generator fails to explore the interesting features of this domain and makes no attempt to incorporate real-world structures into the problems. Also, we feel that too few problems were generated to determine the performance of the planners. Our results show that our problems spanned a whole range of difficulties, whereas the competition problems were found either too hard or too easy. It is our opinion that the results presented here are sufficient to determine the best planner over the whole domain and in addition, provide useful information to the planner designer, regarding the planner's capabilities.

Depth of results: Number and Range

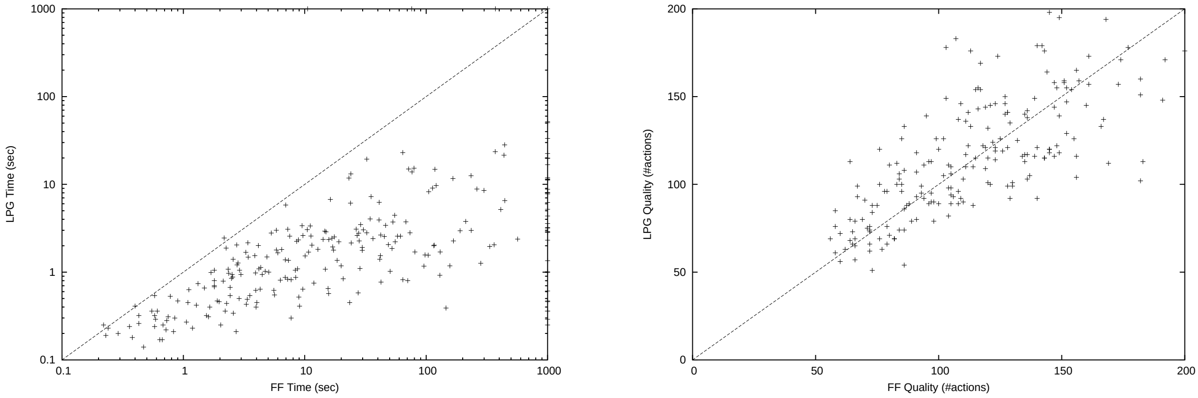
One of the motivations for this work was to improve the quality of results that the planning competition could pro-



(a) Time

(b) Quality

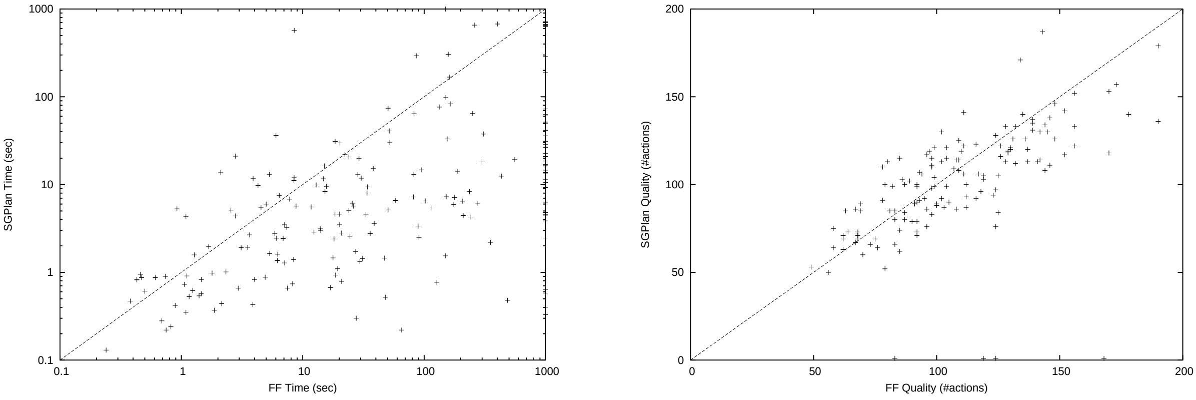
Figure 2: FF vs. LPG Planar Directed Road Network



(a) Time

(b) Quality

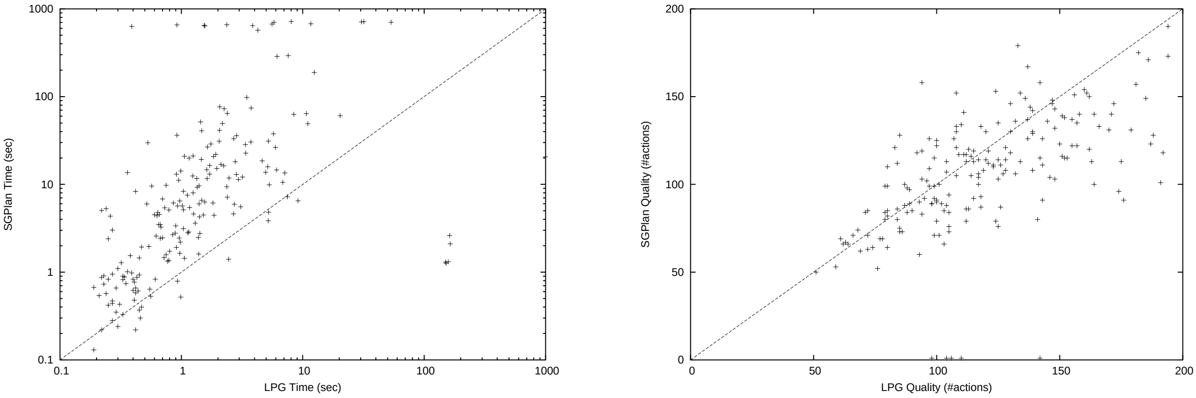
Figure 3: FF vs. LPG Planar Undirected Road Network



(a) Time

(b) Quality

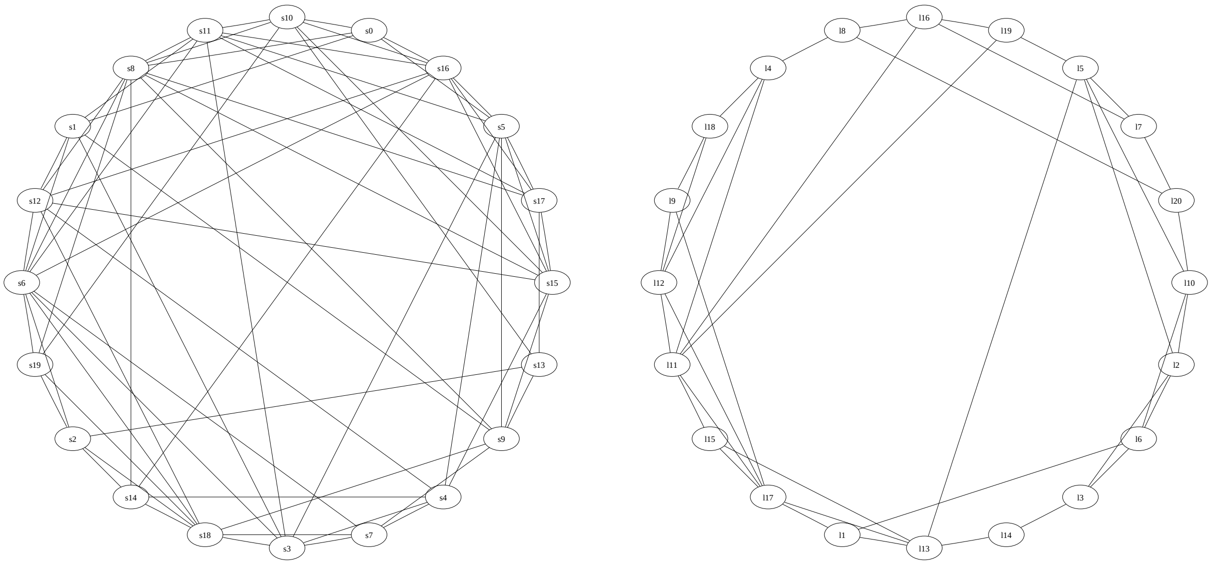
Figure 4: FF vs. SGPlan Planar Directed Road Network



(a) Time

(b) Quality

Figure 5: LPG vs. SGPlan Planar Directed Road Network



(a) Competition

(b) Planar Undirected

Figure 6: Competition Benchmark vs. Planar Undirected Graph Density

vide. We feel that the competition would greatly benefit the community if it not only suggested an overall winner, but also highlighted particular features of planning that individual planners excelled in. The Driverlog domain provides an opportunity to test the planners on many interesting structural problems. However, in the competition only 20 problem instances are generated, hardly enough to make a full exploration of the domain. Table 2 shows the time results for FF, LPG and SGPlan for the competition instances. It is difficult to form any kind of of comparison, as the results are so similar. In contrast, Figure 2 b) shows the time result for FF and LPG for our planar problem set. The large problem set ranging over the entire dimension, provides results that clearly shows how the planners compare throughout an entire range of problem difficulties.

The results that we present for each dimension come from a full range of problem difficulties. We feel that this gives us a strong base to make informed claims about each planner's abilities in terms of these dimensions. In the 2002 competition, the first 15 of the problems for Driverlog provided no challenge to the planners, and the last 5 were all found extremely difficult (mostly impossible) (Long & Fox 2003). The problems failed to provide a smooth range of difficulty. We feel that if claims are going to be made about the quality of plans a planner makes or how quickly it produces those plans, then the planner must have been tested across the whole range of possible problems.

Interesting structure

Driverlog problems have the potential of containing all sorts of structural features. We feel that the dimensions introduced earlier, capture a very interesting selection of these. The competition generator constructs the graphs by randomly forming many connections between nodes, and this results in densely connected graphs. All of the graphs are undirected and the road and path graphs must visit every point. This means that the dimensions that we highlighted either can not, or are very unlikely to appear in any of the problems generated for the competition. The competition therefore fails to explore much of the interesting structure possible in this domain.

Our generators cover several structural features; the problems therefore test the planners across these features. This means that our results can be used to determine more than just the best planner: they also identify how a planner performs on problems with a particular structural feature. In the results section, we identified the dead-end feature as a particular problem for FF, LPG and SGPlan. We feel that this sort of information will provide invaluable feedback to the planner designer, allowing them to focus their research on the areas of weak performance. As discussed, it is unlikely that the competition generator will provide many problems with interesting structure. As a result, it is impossible to identify when a planner performs poorly using the competition instances.

Density and realism

The planning competition is a force that directs the planning community and in our opinion it should be used to push planning towards dealing with real-world situations. Although current planners cannot deal with large real-world problems, we feel that realistic structures should be incorporated into planning problems wherever possible. The road connections in real-world transport network often form planar graphs. As we described previously, the competition Driverlog generator is likely to generate very dense graphs, contrasting with the real model. Figure 6 a) highlights the connectivity of a typical competition problem, where b) shows the more realistic, sparse structure generated by our planar graph generator. The dimensions that we have presented in this work, have been designed specifically to test planners on real-world structural features. It is therefore our opinion that our generator is more likely to include realistic structures within the problems it generates.

Future Work

This short study aims to motivate researchers to take the problem of instance generation more seriously. To further this work, several things can be done:

Create More Generators Driverlog is just one domain from many previous competition domains. Instance generators for the full range of competition domains would help to further refine where planning technology's strengths and weaknesses are.

Complete Driverlog Generator Even the Driverlog generators as described in this work are not complete. New interesting dimensions may be identified, which would require extending the generator to create problems across this new dimension. One of the current dimensions (amount of symmetry) is not yet varied explicitly in the generators. Adding this capacity is part of the future work for this project.

Richer Modelling Language PDDL is capable of expressing far more than the propositional instances generated by our current generator. In the IPC, numeric and temporal versions of Driverlog were tested alongside the purely propositional forms of the problem. These included durations on each of the actions, and also fuel costs for driving. They also had different metrics to optimise. Clearly expanding the generators to these dimensions is essential to further planning technology in these areas.

Real-world Derived Instances Real logistics problems are different from typical Driverlog instances both in size and structure. Real logistics problems have huge numbers of locations. The structure of their underlying maps will remain constant: road networks rarely change significantly. If one goal of the planning community is to address real-world problems, then real-world benchmarks are required. Techniques to exploit structures that are constant between different instances could be developed to tackle these problems.

A Generator Generator There are common, repetitive structures that occur in different planning domains. For instance, there are many problems similar to Driverlog, in which movement across a graph is required. If these structures can be identified, then the dimensions identified here that relate to graph structures could be used as generic dimensions in other problems with similar structures. Therefore, if enough different structures could be identified, then a generic problem generator could be created which would be able to generate instances of any domain that have interesting structure.

Conclusions

In this paper, we have tried to show that the problem of instance generation is of critical importance to the planning community. Having complex domains is not enough. To test planners effectively, then benchmarks that explore all possible structural dimensions of our domains have to be created.

We have identified several structural dimensions for the Driverlog domain, and have created instance generators that explore several of these. After creating many instances our results show that, for the planners tested, there is little difference in plan quality. The planners also cannot handle resource allocation intelligently (as seen in the dead-end example).

We have shown that the IPC generator does not generate structurally interesting instances, and have made various criticisms of the competition benchmarks. It must be remembered that running the IPC already requires a great deal of work, and so this work is not created to undermine the efforts of the organisers. However, it does show that creating instance generators should not simply be the responsibility of the competition organisers.

This work is still preliminary, and a completely unified Driverlog generator that can generate instances anywhere in the structural dimensions is essential. There is still plenty work to be done to understand what structural properties underlie difficult instances. Hopefully this work will convince its readers that instance generation is an important topic both for comparing our planners and for understanding what makes a difficult planning problem.

References

Chen, Y.; Wah, B. W.; and Hsu, C. 2006. Temporal Planning using Subgoal Partitioning and Resolution in SGPlan. *Journal of Artificial Intelligence Research* 26:323–369.

Gerevini, A., and Serina, I. 2002. LPG: A Planner Based on Local Search for Planning Graphs with Action Costs. In *AIPS*, 13–22.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*.

Koehler, J. 1999. RIFO within IPP. Technical report, Albert-Ludwigs University at Freiburg.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of AI Research* 20:1–59.

Slaney, J., and Thiébaux, S. 2001. Blocks world revisited. *Artif. Intell.* 125(1-2):119–153.

VLEPpO: A Visual Language for Problem Representation

Ourania Hatzi[1]**, Dimitris Vrakas**[2]**, Nick Bassiliades**[2]**, Dimosthenis Anagnostopoulos**[1] **and Ioannis Vlahavas**[2]

[1]Harokopio University of Athens, Athens, Greece
{raniah, dimosthe}@hua.gr
[2]Dept. Of Informatics, Aristotle University Of Thessaloniki, Thessaloniki, 54124, Greece
{dvrakas, nbassili, vlahavas}@csd.auth.gr

Abstract

AI planning constitutes a field of interest as its techniques can be applied to many areas. Contemporary systems that are being developed deal with certain aspects of planning and focus mainly on dealing with advanced features such as resources, time and numerical expressions. This paper presents VLEPpO, a Visual Language for Enhanced Planning problem Orchestration. VLEPpO is a visual programming environment that allows the user to easily define planning domains and problems, acquire their PDDL representations, as well as receive solutions, utilizing web services infrastructure.

1. Introduction

AI planning has been an active research field for a long time, and its applications are manifold. A great number of techniques and systems have been proposed during this period in order to accommodate designing and solving of planning domains and problems. In addition, various formalisms and languages have been developed for the definition of these domains, with Planning Domain Definition Language (PDDL) [4][5][6] being dominant among them.

Research among contemporary planning systems has revealed a lack of appropriate integrated visual environments for representing accurately PDDL elements and structures, and consequently using these structures to produce quality plans. This provided the motivation for the work presented in this paper.

The proposed visual tool is intended to cover the need for such an environment by providing an easy to use, efficient graphical user interface, as well as interoperability with planning systems implemented as web services. The elements offered in the interface correspond to PDDL elements and structures, making the representation of most contemporary planning domains possible. Furthermore, importing from and exporting to PDDL features are provided as well. Drag and drop operations along with validity checks make the use of the environment easy even for users not particularly familiar with the language.

The rest of the paper is organised as follows: Section 2 reviews related work in the field by presenting several planning systems, while Section 3 discusses the eminent formalisms for representing planning domains and problems. Section 4 presents our visual tool and demonstrates its use through examples, and finally, Section 5 concludes and discusses future goals.

2. Related Work

There have been a few experimental efforts to construct general-purpose tools which offer user interfaces for defining planning domains and problems, as well as executing planners which provide solutions to the problems.

The GIPO system [1] is based on an object-centric view of the world. The main idea behind it is the notion of change in the state of objects throughout plan execution. Therefore, the domains are modelled by describing the possible changes to the objects existing in the domain. The GIPO system is designed to work with both classical and HTN (Hierarchical Task Networks) domains. In both cases, it offers graphical editors for domain creation, planners, animators for the derived plans and validation tools. The domain models are represented mainly in an internal representation language called OCL (Object Centered Language) [8], which is, as the name implies, object oriented, in accordance with the GIPO system. Translators from and to PDDL have been developed, which cover only a few parts of the language (typed / conditional PDDL).

SIPE-2 [2] is another system for interactive planning and execution of the derived plans. As it is designed to be performance-oriented, it embodies many heuristics for increased efficiency. Another useful feature is the plan execution monitoring, which enables the user to feed new information to the system in case there is some change in the world. In addition, the system offers graphical interfaces for knowledge acquisition and representation, as well as plan visualization. SIPE-2 is an elaborate system with a wide range of capabilities. However, it uses the ACT formalism, which is quite complicated and does not correspond directly to PDDL, although PDDL descended partially from this formalism, but also from other formalisms such as ADL. Therefore, there is no way to easily use a PDDL file to construct a domain in SIPE-2, or export the domain or problem to PDDL.

ASPEN is an environment for automated planning and scheduling. It is an object-oriented system, originally targeted to space mission operations. Its features include an expressive constraint modelling language which is used for defining the application domain, systems for defining activity requirements and resource constraints, as well as temporal constraints. In addition, a graphical user interface is included, but its use in confined to visualization of plans and schedules, in systems where the problem solving process is interactive.

ASPEN was developed for the specific purposes of space mission operations and therefore, it has only a few vague correspondences to PDDL. Furthermore, it does not offer a graphical interface for creating the planning domains.

In conclusion, although the above systems are useful, none of them offers direct visual representation of PDDL elements, a feature which would make the design very efficient for the users already familiar with the language. Moreover, even the systems which offer translation to PDDL do not cover important features of the language. It should be mentioned that a couple of other systems which provide user interfaces can be found in the literature, but they are not mentioned in this section because of their being developed for specific purposes.

The VLEPpO tool is based on ViTAPlan [3] a visualization environment for planning based on the HAP_{RC} planning system. VLEPpO extends ViTAPlan in numerous ways providing the user with visualization capabilities for most of the advanced features of PDDL [6] and a more accurate and expressive visual language.

3. Problem Representation

A crucial step in the process of solving a search problem is its representation in a formal language. The choice of the language can significantly affect not only the comprehensiveness of the representation but also the efficiency of the solver. The PDDL language is nowadays the standard for representing planning problems. PDDL is partially based on the STRIPS [7] formalism. Since the environment presented in this work has a close connection with PDDL, a brief description of the most important language elements will be provided in the following section.

3.1. The PDDL Definition Language

PDDL [4] stands for Planning Domain Definition Language. Although it was initially designed for planning competitions such as AIPS and IPC, it has become a standard in the planning community for modelling planning domains. PDDL focuses on expressing the physical properties of the domain at hand in each planning problem, such as the available predicates and actions. However, at the same time, there are no structures or elements in the language to provide the planner with advice, that is, guidelines about how to search the solution space, although extended notation may be used, depending on the planner.

Each domain definition in PDDL consists of several declarations, which include types of entities, variables, constants, literals that are true at all times called timeless, and predicates. In addition, there are declarations of actions, axioms and safety constraints. These elements are explained in the following paragraphs.

Variables have the same semantics as in any other definition language, and are used in conjunction with built-in functions for expression evaluation. In more recent versions of PDDL, fluents seem to gain momentum instead of variables when there is a need for values that can change over time, as a result of an action.

Constants represent objects that do not change values and can be used in the domain operators or the problems associated with a domain.

Relations between objects in the domain are represented by predicates. A predicate may have an arbitrary number of arguments, whose ordering is important in PDDL. Predicates are used to describe the state of the world at a specific moment. Moreover, they are used as preconditions and results of an action.

Timeless predicates are predicates that are true at all times. Therefore, they cannot appear as a result of an action unless they also appear among its preconditions. In other words, timeless predicates are not affected by any actions available to the planner.

Actions enable transitions between successive situations. An action declaration mentions the parameters and variables involved, as well as the preconditions that must hold for the action to be applied. PDDL offers two choices when it comes to defining the results of the action: The results can either be a list of predicates called effects, or an expansion, but not both at the same time. The effects, which can be both conditional and universally quantified, express how the world situation changes after the action is applied. More specifically, inspired by the STRIPS formalism, the effects include the predicates that will be added to the world state and the predicates that will be removed from the world state.

Axioms, in contrast to actions, state relationships among propositions that hold within the same situation. The necessity of axioms arises from the fact that the action definitions do not mention all the changes in all predicates that might be affected by an action. Therefore, additional predicates are concluded by axioms after the application of each action. These are called derived predicates, as opposed to primitive ones. In more recent versions of the language the notion of derived predicates has replaced axioms, but the general idea described remains the same.

Safety constraints in PDDL are background goals which may be broken during the planning process, but ultimately they must be restored. Constraint violations present in the initial situation do not require to be fulfilled by the planner.

After having defined a planning domain, problems can be defined with respect to it. A problem definition in PDDL must specify an initial situation and a final situation, referred to as goal. The initial situation can be specified either by name, or as a list of literals assumed to be true, or a combination of both. In the last case, literals are treated as effects; therefore they are added to the initial situation stated by name. Again, the closed-world assumption holds, unless stated otherwise. Therefore, all predicates which are not explicitly defined to be true in the initial state are assumed to be false. The goal can be either a goal description, using function-free first order predicate logic, including nested quantifiers, or an expansion of actions, or both. The solution given to a problem is a sequence of actions which can be applied to the initial situation, eventually producing the situation stated by the goal description, and satisfying the expansion, if there is one.

PDDL 2.1 [5] was designed to be backward compatible with PDDL 1.2, and to preserve its basic principles. It was developed by the necessity for a language capable of expressing temporal and numeric properties of planning domains.

The first of the extensions introduced were numeric expressions. Primitive numeric expressions are values of functions which are associated with tuples of domain objects. Further numeric expressions can be constructed using primitive ones and arithmetic operators. In order to support numeric expressions, new elements were added to the language. Functions are now part of the domain definition and, as mentioned above, they associate a number of objects with an arithmetic value. Moreover, conditions were introduced, which are in fact comparisons between pairs of numeric expressions. Finally, assignment operations are possible, with the use of built-in assignment operators such as *assign*, *increase* and *decrease*. The actual value for each combination of objects given by the functions is not stated in the domain definition but must be provided to the planner in the problem definition.

A further extension to PDDL facilitated by numeric expressions is plan metrics. Plan metrics specify the way a plan should be evaluated, when a planner is searching not for any plan, but for the optimal plan according to some metric.

Other extensions in this version include durative actions, both discretised and continuous. Up to now, actions were considered instantaneous. Durative actions, as the term implies, have a duration which is declared along with their definition. Furthermore, as far as discretised durative actions are concerned, temporal annotations are introduced to their conditions and effects. A condition can be annotated to hold at the start of the interval, at the end of the interval, or all over the interval during which the action lasts. An effect can be annotated as immediate, that is, it takes place at the start of the interval, or delayed, that is, it takes place at the end of the interval.

In PDDL 3.0 [6] the language was enhanced with constructs that increase its expressive power regarding the plan quality specification. The constraints and goals are divided into strong, which must be satisfied by the solution, and soft, which may not be satisfied, but are desired. In addition, the notion of *plan trajectories* is introduced, which allows the specification of intermediate states that a solution has to reach, before it reaches the final state.

4. The Visual Language

VLEPpO (Visual Language for Enhanced Planning problem Orchestration) is an integrated system for visually designing and solving planning problems, implemented in Java. It offers an efficient and easy-to-use graphical interface, as well as compatibility and interoperability with PDDL. The main goal during the implementation of the graphical component of the tool was to keep the interface as simple and efficient as possible, but, at the same time, represent accurately and flexibly the features of PDDL. The range of PDDL elements that can be represented in the tool is quite wide, and covers the elements that are used more frequently in contemporary planning domains and problems. In the following, the features of the tool will be discussed in more detail.

4.1. The Entity – Relation Model

The entity – relation model is used to design the structure of data in a system. Our visual tool employs this well-known formalism, adapting it to PDDL. Therefore, the entities in a planning domain described in PDDL are the objects of the domain, while the relations are the predicates. These elements are represented visually in the tool by various shapes and connections between them.

A class of objects in the tool is represented visually by a coloured circle. A class in PDDL represents a type of domain objects or action parameters. From a class the user can create parameters of this type in operators, and objects of this type in problems, by dragging and dropping the class on an operator or a problem, respectively. The type of a parameter or object is denoted by their colour, which is the same as the corresponding class.

Consider the gripper domain for example, where there is a robot with N grippers that moves in a space, composed of K rooms that are all connected with each other. All the rooms are modelled as points and there are connections between each pair of points and therefore the robot is able to reach all rooms starting from any one of them with a simple movement. In the gripper domain there are L numbered balls which the robot must carry from their initial position to their destination.

Following a simple analysis the domain described above can be encoded using four classes: robot, gripper, room and ball. However, since the domain does not support the existence of multiple robots, the class robot can be implicitly defined and therefore there is no need for it. The three remaining classes are represented in VLEPpO using three coloured circles as outlined in Figure 1.

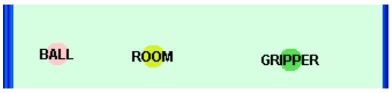


Figure 1. The classes in Gripper domain.

A relation is represented by a coloured rectangle with black outline. A relation corresponds to a domain predicate in PDDL and it is used for defining connections among classes. The relations in PDDL and therefore in VLEPpO are of various arities. Unary relations are usually used to define properties of classes that can be modeled as binary expressions that are either true or false (e.g. closed(Door1)).

If at least one pair of class and relation is present in the domain, the user can add connections between them. Each connection represents an argument of a relation, and the class shows the type of this argument. A relation may have as many arguments as the user wishes, of any type the user wishes. The arguments are ordered according to the numbers on each connection, because this ordering is important to PDDL.

The Gripper domain has four relations, as depicted in Figure 2: a) *at-robot*, which specifies the position of the robot and it is connected only with one instance of room, b) *at* which specifies the room in which each ball resides and therefore is connected with an instance of ball and an instance of room, c) *holding* which defines the alternative position of a ball, i.e. it is held by the robot and therefore it is connected with an instance of ball and an instance of gripper and d) *empty* which is connected only with an instance of gripper and states that the current gripper does not hold any ball.

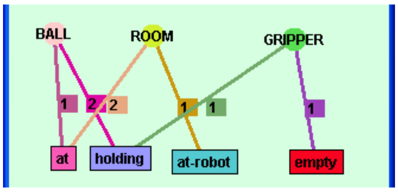


Figure 2. The relations in the Gripper domain.

Note here that although non-typed PDDL requires only the arity for each predicate and not the type of objects for the arguments, the interface obliges the user to connect each predicate with specific object classes and this is used for the consistency check of the domain design. According to the design of Figure 2, the arity of predicate holding, for example, is two and the specific predicate can only be connected with one object of class ball and one object of class gripper.

The aforementioned elements, classes, relations and connections combined together form the entity – relation model of the data for the planning domain the user is dealing with.

4.2. Representing Operators

Operators have direct correspondence to PDDL actions, which enable transitions between successive situations. The main parts of the operator definition are its preconditions and results, as well as the parameters. Preconditions include the predicates that must hold for the action to be applied. Results are the predicates that will be added or removed from the world state after the application of the action. Operators in the visual tool are represented by light blue resizable rectangles in the Operator Editor, comprised by three columns. The left column holds the preconditions, the right column holds the effects, and the middle one the parameters.

Dragging and dropping a relation on an operator will add the predicate to the preconditions or effects, depending on which half of the operator the shape was dropped on. Parameters can be created in operators by dropping classes on them. Adding a connection in the ontology enables the user to add corresponding connections in the operators. Other elements that can be imported in operators will be discussed in more detail in the section about advanced features.

For example, in the gripper domain there are three operators: a) *move* which allows the robot to move between rooms, b) *pick* which is used in order to lift a ball using a gripper and c) *drop* which is the direct opposite of pick and is used to leave a ball on the ground (Figure 3)

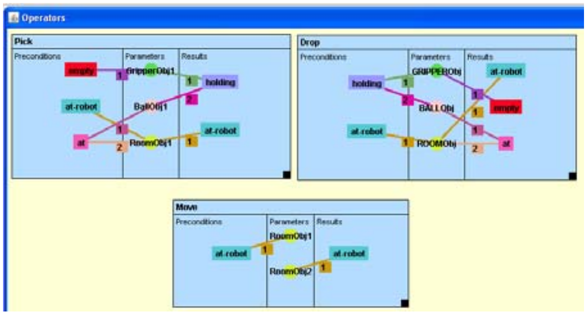


Figure 3. The operators in the Gripper domain.

The default view for an operator is in preconditions / results view which follows a declarative schema that is different from the classical STRIPS/PDDL approach. However, there is a direct way to transform definitions from one approach to the other.

Although the preconditions / results view is more appropriate for visualizing operators, the system gives the user the option to use the classical add / delete lists view, therefore the STRIPS formalism is accommodated as well. If selected, the column on the left, as before, shows the preconditions that must hold for the action to be executed, but the column on the right shows the facts that will be added and deleted from the current state of the world upon the execution of the action.

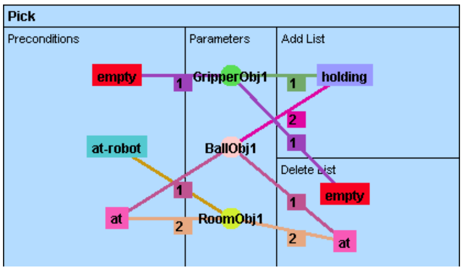


Figure 4. Pick operator in add/delete lists view.

As an example, the *pick* operator of the Gripper domain is considered. According to the STRIPS formalism, the operator is defined by the following three lists, also depicted in Figure 4.

prec = {empty(GripperObj1), at-robot(RoomObj1),
at(BallObj1,RoomObj1)}
add = {holding(GripperObj1, BallObj1)}
del = {empty(GripperObj1), at(BallObj1, RoomObj1)}

The equivalent operator in Preconditions / Results view is presented in Figure 5.

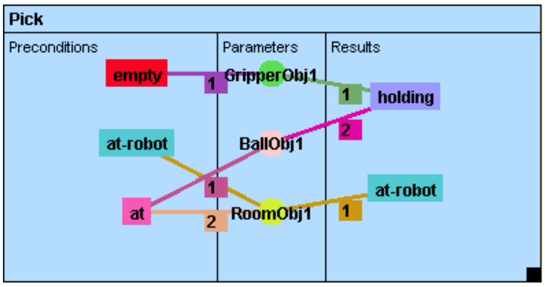


Figure 5. Pick operator in preconditions / results view.

4.3. Representing Problems

For every domain defined in PDDL a large number of problems that correspond to this domain can also be defined. Problem definitions state an initial and a goal situation, and the task of a planner is to find a sequence of operators that, if applied to the initial situation, will provide the goal situation. The problem shape in the visual tool is much like an operator in form, but different semantically. It is represented by a three-column resizable rectangle in the Problem Editor. Left column holds the predicates in the initial state, right column holds the predicates in the goal state, and middle column holds the objects that take part in the problem definition.

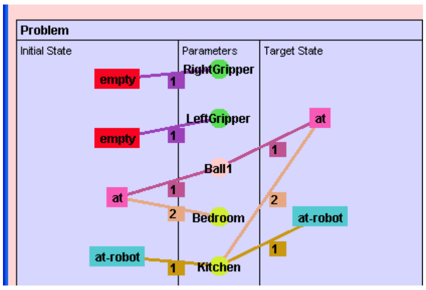


Figure 6. A Problem instance of the Gripper domain.

Figure 6 presents a problem instance of the gripper domain, which contains two rooms (Bedroom and Kitchen), one ball (Ball1) and the robot has two grippers (rightGripper and leftGripper). The initial state of the problem defines the starting locations of the robot and the ball (Kitchen and Bedroom respectively) and that both grippers are free. The goals specify that the destination of both the ball and the robot is the kitchen.

4.4. Advanced Features

The basic PDDL features described above are adequate for simple planning domains and problems. However, the language has many more features divided into subsets referred to as requirements. An effort has been made in order for the visual tool to embody the most significant and frequently used among them.

An advanced design element offered by the system, which has direct representation in PDDL, is a constant. The constant is visually represented similarly to a class, but it is enhanced with a red circle around it to discriminate it from a class. The constant must be of a type, and the tool enables the user to drag and drop it on a class to denote that. Constants can be used either in an operator or in a problem, where they are treated similarly to parameters or objects, respectively.

A derived predicate is another advanced PDDL feature that is represented by a group of design elements in the visual tool. The term refers to predicates that are not affected by operators, but they are derived by other relations using a set of rules. Derived predicates in fact existed in the first version of the PDDL language as well, under the notion of *axioms*. Visually, they are represented by a rounded rectangle with a specific colour, but they are not complete unless they are enhanced with an AND/OR tree that indicates the way they are derived by other relations. Consequently, AND, OR and NOT nodes for the construction of the tree are also offered as design elements. In the current implementation, AND and OR nodes are binary, that is, they accept only two possible arguments, while NOT nodes are by default unary. Each of the node arguments can be either another node of any type, or a relation. An example of a derived predicate is depicted in Figure 7.

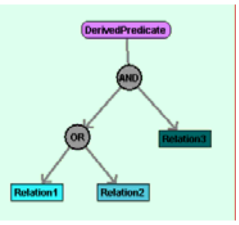


Figure 7. A derived predicate with AND/OR tree.

Among the advanced features is the option to indicate that a predicate is timeless, that is, the predicate is true at all times. This operation involves a lot of validity checks, which will be explained in the corresponding paragraph.

Another PDDL feature incorporated in the tool are numerical expressions. In order for numerical expressions to function properly, the definition of a number of other elements is involved. Consequently, a combination of design elements in each frame is used. First of all, in the ontology frame the user can import functions, which are represented by rectangles with double outline. These functions may or may not have arguments. As with simple relations, functions can be dragged on operators. However, in order to appear in the PDDL description of an operator, they must be involved in a condition or in an assignment. The next step is to actually import conditions and assignments which involve these functions in the operator. In that case, a dialog box appears facilitating the import of a condition or an assignment, by showing all the available options that the user can select among. Furthermore, for each function imported in the tool, a new rectangle appears in the problem frame, which corresponds to this function. This rectangle is used to declare the initial values of the function for the problem at hand.

Furthermore, the system supports discretised durative actions. The definition of a durative action includes setting the duration of an operator, in combination with temporal annotations (Figure 8). In this case, the action is considered to last a specific period of time, while the preconditions can be specified to hold at the beginning of this period, at the end of this period, or all over the period (combination of these choices is also possible). Effects can be immediate, that is, happen at the beginning of the action, or delayed, that is happen at the end of the action.

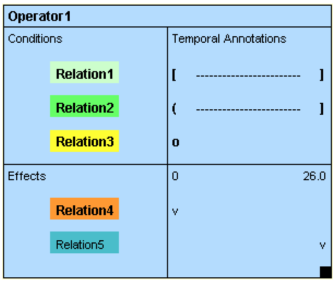


Figure 8. An example of a durative action.

Finally, a very useful element for problem designing is maps. Maps represent a special kind of relations that have exactly two arguments of the same type, and are expected to have many instances in the initial state of a problem (Figure 9). For each relation that fulfills these conditions a map can be created. Objects which take part in the instances of the relation can then be dragged on the map, and connections can be created between them. Each of these connections represents an instance of the relation that the map corresponds to. In conclusion, maps do not have an exact representation to PDDL, but they express a part of the initial state of the world, thus making the problem shape more readable. The use of maps is not mandatory, as the same relations can be simply represented in the problem shape.

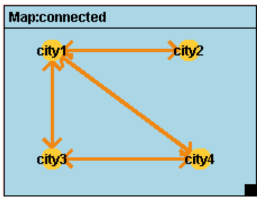


Figure 9. A map for the relation connected(C1, C2).

4.5. Syntax and Validity Checking

A very important aspect in every tool for designing and editing planning domains is syntax and validity checking. Planning domains have to be checked for consistency within their own structures, and planning problems have to be checked for consistency and correspondence to the related domains. This visual tool attempts to detect inconsistencies at the moment they are created and notify the user about them, before they propagate in the domain. In the remainder of this paragraph several examples will be given, in order to illustrate the validity checking processes of the system.

Whenever the user attempts to insert a new connection in an operator or in a problem, necessary checks are performed and if a corresponding connection cannot be found in the ontology an appropriate error message is shown. Special care must be taken to verify that the types of parameters and objects match to the types of arguments of the predicates.

As already mentioned, the system supports timeless predicates, which are, by definition, true at all times. Therefore, they are allowed to appear in the preconditions of an operator, but not in the add or delete lists. As a consequence, if the user tries to add a timeless predicate in the preconditions part of an operator, it will automatically appear in the effects part, so the add and delete lists will not be affected. Furthermore, if the user tries to set a predicate timeless, checks will be performed to determine if this operation is allowed. Finally, timeless predicates are not allowed to appear in a problem. In all above cases, error messages occur in order to warn the user and help them correct the domain inconsistencies.

Another example is that of constants. Checks are performed to confirm that the class of a constant has already been defined before the user attempts to use the constant in an operator or a problem. Furthermore, additional checks are performed about the types of arguments, similar to those performed for simple objects.

4.6. Translation to and from PDDL

The capability to export the domains and problems designed in the tool to PDDL constitutes another important feature. All of the design elements that the user has imported in the domain, such as predicates and operators, along with comments, are exported to a PDDL file, which is enhanced with the appropriate requirements tag. The user is offered the option to use typing, therefore, the same domain can produce two different PDDL files, one with the *:typing* requirement and one without it. Details about exporting are presented in the remainder of the paragraph.

Despite the fact that a class in the visual tool always represents the same notion, that is, the type of domain objects or parameters, it takes different forms when it comes to exporting the domain. In case the requirement *:typing* is declared, the class name is included in the (:types) construct of the domain definition, and for each object, parameter and constant a type must be declared. In case typing is not used, classes are treated as timeless unary predicates and appear in the corresponding part of the domain definition. In addition, for each parameter in an operator, a precondition that denotes the type of the parameter must be added in the PDDL definition, although it does not appear visually in the tool. Likewise, for each object, a new initial literal denoting the type of this object must be included in the problem definition.

The elements in the Ontology Editor are combined together in order to formulate the domain constructs in the syntax that the language imposes. For example, relations, connections and, if typing is used, classes are combined to formulate the predicates construct. Likewise, functions and derived predicates constructs are formed. As far as constants are concerned, they may appear in the place of parameters in operators and objects in problems, and they also appear in the special construct (:constants) in the domain definition.

Exporting the operators is quite more complicated, because a combination of several elements of the Operator Editor and the Ontology Editor is needed. Slight changes occur to an operator definition depending on whether the *:typing* requirement is declared.

Finally, exporting the problems is quite similar to exporting the operators, but the problems are stored in a different PDDL file. Therefore, numerous problems can be defined for the same domain. If maps are used, care must be taken to include the information they embody in the list of predicates included in the initial state. Furthermore, if functions are used, their initial values provided by the user in the Problem Editor will be part of the declaration of the initial state of the problem, in the corresponding construct.

The visual tool also offers the feature of importing planning domains and problems expressed in PDDL, visualizing them, and thus enabling the user to manipulate them. However, importing PDDL is subject to some restrictions. The most important is that the domains and problems must declare the *:typing* requirement. If no typing is used, syntax is not enough, and semantic information is necessary in order to discriminate types of objects from common unary predicates.

4.7. Interface with Planning Systems

As the tool is intended to be an integrated system not only for designing but for solving planning problems as well, an interface with planning systems is necessary. This is achieved by providing the ability to discover and communicate with web services which offer implementations of various planning algorithms. Therefore, a dynamic web service client has been developed as a subsystem. The requirement for flexibility in selecting and invoking a web service justifies the decision to implement a dynamic client instead of a static one. Therefore, the system can exploit alternative planning web services according to the problem at hand, as well as cope with changes in the definitions of these web services.

The communication with the web services is performed by means of exchanging SOAP messages, as the web service paradigm dictates. However, in a higher level, the communication is facilitated by the use of the PDDL language, which constitutes the common ground between the visual tool and the planners. An additional advantage of using PDDL is that the visual tool is released by the obligation to determine the PDDL features that a planner can handle, thus leaving each planning system to decide for itself.

The employment of web services technology for implementing the interface results in the independency of the visual tool from the planning or problem solving module. Such a decoupling is essential since new planning systems which outperform the current ones are being developed. Each of them can be exposed as a web service and then invoked for solving a planning problem without any further changes to the visual tool or the domains and problems already designed and exported as PDDL files.

5. Conclusions and Future Work

In this paper a visual tool for defining planning domains and problems was proposed. The tool offers an efficient user interface, as well as interoperability with PDDL, the standard language for planning domain definition. The elements represented in the tool cover a wide range of the language, while the user is significantly facilitated by the validity checks performed during the design process. The use of the tool is not confined to designing planning problems, but the ability to solve them by invoking planners implemented as web services is offered as well. Therefore, the tool is considered an integrated system for designing and solving planning problems.

Our future goals include the extension of the tool in order to represent even more complex PDDL language elements, as well as other planning approaches, such as HTN (Hierarchical Task Network) planning. Such an extension is believed to broaden the range of real world problems that can be represented and solved by the tool. Visual representation of the produced plans, along with plan metrics are also among our imminent goals.

Acknowledgements

This work was partially supported by a PENED program (EPAN M.8.3.1, No. 03EΔ73), jointly funded by the European Union and the Greek government (General Secretariat of Research and Technology).

References

[1] T. L. McCluskey, D. Liu, Ron M. Simpson, “GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment”, International Conference on Automated Planning and Scheduling (ICAPS), 2003

[2] Wilkins, D. E., Lee, T. J. and Berry, P., Interactive Execution Monitoring of Agent Teams, Journal of Artificial Intelligence Research, 18 (2003), pp. 217-261.

[3] D. Vrakas, I. Vlahavas, “A Visualization Environment for Planning”, International Journal on Artificial Intelligence Tools”, Vol. 14 (6), 2005, pp. 975-998, World Scientific.

[4] Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D. and Wilkins, D., "PDDL -- the planning domain definition language". Technical report, Yale University, New Haven, CT (1998).

[5] Fox, M. and Long, D., "PDDL2.1: An extension to PDDL for expressing temporal planning domains". Journal of Artificial Intelligence Research, 20 (2003), 61-124.

[6] Gerevini, A. and Long, D., "Plan Constraints and Preferences in PDDL3", Technical Report R.T. 2005-08-47, Department of Electronics for Automation, University of Brescia, Italy.

[7] Fikes, R. and Nilsson, N. J., STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence*, Vol 2 (1971), 189-208.

[8] Liu, D., and McCluskey, T. L. 2000. The OCL Language Manual, Version 1.2. Technical report, Department of Computing and Mathematical Sciences, University of Huddersfield

Constraint Programming Search Procedure for Earliness/Tardiness Job Shop Scheduling Problem

Jan Kelbel and **Zdeněk Hanzálek**
Centre for Applied Cybernetics, Department of Control Engineering
Czech Technical University in Prague, Czech Republic
{kelbelj,hanzalek}@fel.cvut.cz

Abstract

This paper describes a constraint programming approach to solving a scheduling problem with earliness and tardiness cost using a problem specific search procedure. The presented algorithm is tested on a set of randomly generated instances of the job shop scheduling problem with earliness and tardiness costs. The experiments are executed also for three other algorithms, and the results are then compared.

Introduction

Scheduling problems with storage costs for early finished jobs and delay penalties for late jobs are common in industry. This paper describes a constraint programming (CP) approach (Barták 1999) to solve a scheduling problem with earliness and tardiness costs, which is for distinct due dates NP-complete already on one resource (Baker & Scudder 1990).

This paper focuses on the job shop scheduling problem with earliness and tardiness costs. This problem—introduced in (Beck & Refalo 2001; 2003)—is solved there using hybrid approach based on probe backtrack search (El Sakkout & Wallace 2000) with integration of constraint programming and linear programing. This hybrid approach performed significantly better than the generic (naive) CP and MIP algorithms. With another hybrid approach, combining local search and linear programming (Beck & Refalo 2002), results slightly worse than in (Beck & Refalo 2001) were obtained. The large neighborhood search (Danna & Perron 2003) applied to the same earliness tardiness job shop problem outperformed both hybrid approaches of Beck & Refalo.

This paper describes a search procedure for scheduling problems with earliness and tardiness costs which initially tries to assign to variables those values that lead to a solution with minimal cost. It is developed by improving of the search procedure used in (Kelbel & Hanzálek 2006) where constraint programming is applied to an industrial case study on a lacquer production scheduling. While in (Kelbel & Hanzálek 2006) tardy jobs were not allowed, the procedure described in this paper allows both early and tardy jobs, i.e. optimal solutions are not discarded.

The proposed search procedure is tested on a set of randomly generated instances of the job shop scheduling problem with earliness and tardiness costs. It significantly outperforms simple (default) models introduced in (Beck & Refalo 2003), and in average it gives results better than the Unstructured Large Neighborhood Search (Danna & Perron 2003).

Earliness Tardiness Job Shop Scheduling Problem

The definition of the earliness tardiness job shop scheduling problem (ETJSSP) is based on (Beck & Refalo 2003). We assume a set of jobs $\mathcal{J} = \{J_1, \dots, J_n\}$ where job J_j consists of a set of tasks $\mathcal{T}_j = \{T_{j,1}, \dots, T_{j,n_j}\}$. Each task has given processing time $p_{j,i}$, and required dedicated unary resource from a set $\mathcal{R} = \{R_1, \dots, R_m\}$. Starting time $S_{j,i}$ of a task, and completion time defined as $C_{j,i} = S_{j,i} + p_{j,i}$, determine the result of the scheduling problem. For each job J_j there are precedence relations between tasks T_i and T_{i+1} such that $C_{j,i} \leq S_{j,i+1}$ for all $i = 1, \dots, n_j - 1$, i.e. $\mathcal{T}_j$, the set of tasks, is ordered.

Concerning earliness and tardiness costs, each job has assigned a due date d_j, i.e. the time when the last task of the job should be finished. In general, the due dates are distinct. The cost function of the job J_j is defined as $\alpha_j(d_j - C_{j,n_j})$ for early job and $\beta_j(C_{j,n_j} - d_j)$ for tardy job, where α_j and β_j are earliness and tardiness costs of the job per time unit. Taking into account both alternatives, the cost function of the job can be expressed as

$$f_j = \max(\alpha_j(d_j - C_{j,n_j}), \beta_j(C_{j,n_j} - d_j)). \qquad (1)$$

An optimal solution of the ETJSSP is the one with minimal possible sum of costs over all jobs

$$\min \sum_{J_j \in \mathcal{J}} f_j.$$

In this article, a specific ETJSSP will be considered in order to be consistent with the original problem instances (Beck & Refalo 2003). All jobs have the sets of tasks with the same cardinality, which is equal to the number of resources, i.e. $n_j = m$ for all j. Each of the n_j tasks of the job is processed on a different resource. Next, the problem has a work flow structure: the set of resources $\mathcal{R}$ is partitioned into two disjunctive sets $\mathcal{R}_1$ and $\mathcal{R}_2$ of about the same cardinality, and the tasks of each job must use all resources from the first set before any resource from the second set, i.e. task $T_{j,i}$ for all $i = 1, \ldots, |\mathcal{R}_1|$ requires resource from set $\mathcal{R}_1$, and task $T_{j,i}$ for all $i = |\mathcal{R}_1| + 1, \ldots, n_j$ requires resource from set $\mathcal{R}_2$.

The Model With Search Procedure for ETJSSP

When solving constraint satisfaction problems (Barták 1999), constraint programming systems employ two techniques—constraint propagation and search. The search consists of a search tree construction by a search procedure (called also a labeling procedure) and applying a search strategy (e.g. depth-first search) to explore the tree. The search procedure typically makes decisions about variable selection (i.e. which variable to choose) and about value assignment (i.e. which value from domain to assign to the selected variable).

Our approach to solving ETJSSP is based on usual constraint programming model with a problem specific search procedure. The scheduling problem is modeled directly by using a formulation from the previous section, yet by using higher abstraction objects for scheduling (e.g. tasks and resources) available in ILOG OPL Studio (ILO 2002). The model uses scheduling-specific edge-finding propagation algorithm for disjunctive resource constraints (Carlier & Pinson 1990). In the used CP system we obtained better performance of the computations when the cost function (1) was expressed as $f_j \geq \alpha_j(d_j - C_{j,n_j}) \wedge f_j \geq \beta_j(C_{j,n_j} - d_j)$.

Most of the constraint programming systems have a default search procedure that builds the search tree by assigning the values from domains to variables in increasing order. The idea of our search procedure is based on the fact that only C_{j,n_j}, the completion time of the last task of the job, influences the value of the cost function, and that the values of C_{j,n_j} inducing the lowest values of cost functions f_j should be examined first.

The search procedure, inspired by time-directed labeling (Van Hentenryck, Perron, & Puget 2000), is directed by the cost, only once at the beginning of the search (as an initialization of the search tree), however. It is denoted as cost-directed initialization (CDI) and performs as described in Algorithm 1: variables representing completion time C_{j,n_j} are selected in increasing order of the size of their domains, then the value selection is made according to the lowest value possible of the cost function. In the second branch of the search tree, this value is disabled. This is done only once for each task T_{j,n_j}, then the search continues with the default search procedure.

Slice Based Search available in (ILO 2002), based on (Beck & Perron 2000), and similar to Limited Discrepancy Search (Harvey & Ginsberg 1995) is used as a search strategy to explore the search tree constructed by the CDI procedure. This is necessary for obtaining good performance, since using depth first search instead, the algorithm was not able to find any solution for about 50% of larger size instances of the ETJSSP.

Algorithm 1 – CDI search procedure

```
1. sort the last tasks of all jobs, $T_{j,n_j}$ for all $j$, according to the nondecreasing domain size of $C_{j,n_j}$
2. for each task from the sorted list from domain of $C_{j,n_j}$ select a value $v_j$ leading to minimal $f_j$ and create two alternatives in the search tree:
   • $C_{j,n_j} = v_j$
   • $C_{j,n_j} \neq v_j$
3. Continue with the default search procedure for all variables
```

Experimental Results

The proposed algorithm CDI was tested against two simple generic models introduced in (Beck & Refalo 2003), a mixed integer programming model with disjunctive formulation of the problem (MIP), and a constraint programming model with *SetTimes* heuristic as a search procedure and depth-first search as a search strategy (ST). The third model used for performance comparison is the Unstructured Large Neighborhood Search (uLNS) (Danna & Perron 2003) by enabling Relaxation Induced Neighborhood Search (RINS) via `IloCplex::MIPEmphasis=4` switch in Cplex 9.1 (Danna, Rothberg, & Le Pape 2005; ILO 2005), while using the same MIP model as in (Beck & Refalo 2003). The hybrid algorithm from (Beck & Refalo 2003) was not used due to its implementation complicacy.

Benchmarks are randomly generated instances of the ETJSSP according to Section 6.1 in (Beck & Refalo 2003). The problem instances have a work flow structure. Processing times of tasks are uniformly drawn from the interval $[1, 99]$. Considering the lower bound tlb of the makespan of the job shop according to (Taillard 1993), and a parameter called looseness factor lf, the due date of the job was uniformly drawn from the interval $[0.75 \cdot tlb \cdot lf, 1.25 \cdot tlb \cdot lf]$. The job shops were generated for three $n \times m$ sizes, 10×10, 15×10, and 20×10, and for $lf \in \{1.0, 1.3, 1.5\}$. Twenty instances were generated for each lf—size combination.

The tests were executed using ILOG OPL Studio 3.6 with ILOG Solver and Scheduler for the CP models, and ILOG Cplex 9.1 for the MIP models, all running on a PC with CPU AMD Opteron 248 at 2.2 GHz with 4 GB of RAM. The time limit for each test was 600 s, after which the execution of the test computation was stopped, and the best solution so far was returned.

Table 1 shows the average ratio of the costs of the best solutions obtained by the MIP, uLNS, and ST to the best solutions obtained by CDI, for all types of instances.

In Tables 2 and 3 the ST algorithm will not be included due to its poor performance. Table 2 shows the number of instances solved to optimality within 600 s time limit, and also the number of instances, for which the algorithm proved the optimality of the solution. The CDI usually needed less time than the MIP or uLNS to find a solution with optimal cost, but in many cases it was not proven as an optimum in given time or memory limit. In Table 2 a solution found by the CDI model was considered as the optimal solution when

size	10×10			15×10			20×10		
lf	MIP/CDI	uLNS/CDI	ST/CDI	MIP/CDI	uLNS/CDI	ST/CDI	MIP/CDI	uLNS/CDI	ST/CDI
1.0	1.8	1.2	2.6	4.7	3.1	6.2	5.3	4.9	6.7
1.3	4.8	1.8	9.2	18.4	5.3	28.3	14.0	14.3	25.8
1.5	3.8	2.1	8.1	7.9	1.9	37.9	5.5	5.7	50.6

Table 1: Average ratio for the best values of cost functions of solutions found within 600 s

the value of the objective function was equal to the one of the proven optimal solution found by the MIP models or to a lower bound found by the MIP.

Table 3 is inspired by (Beck & Refalo 2002). For each problem instance, the lowest cost obtained by any of the used algorithms is selected. Then, Table 3 contains the number of instances for which the algorithm found the solution with the best cost, i.e. equal to the lowest cost, and the number of solutions with uniquely best cost, i.e. if no other algorithm has found solution with the same or lower cost.

Conclusion and Future Work

We have shown an algorithm called cost-directed initialization (CDI) designed to solve the earliness-tardiness scheduling problem. The algorithm was compared to other algorithms MIP, uLNS, and ST, on randomly generated earliness tardiness job shop benchmarks. The CDI was able to find within 600 s a solution that is usually better than the one found by any of the MIP, uLNS, or ST. With respect to the best obtained value of the cost function, the CDI algorithm performed better than the other algorithms. However, the weak point of the CDI is that the optimum, even if it is found, is usually not proved.

Since the CDI search procedure does not exploit the structure of the job shop problem, it is possible to apply it on other earliness/tardiness problems but the results may vary. Revisiting the lacquer production scheduling problem (Kelbel & Hanzálek 2006) with the CDI, the solution of the case study was further improved from the cost 886,535 to 777,249 due to the allowance of tardy jobs.

The earliness tardiness job shop scheduling problem, as considered in this paper, does not fully correspond to real production, since only the last tasks of jobs have direct impact on the cost of the schedule. If there is enough time, i.e. the looseness factor is big, there can be quite a big delay between the tasks of the same job, and so a storage would be needed also during the production, but at no cost (since no such cost is defined). So the payed storage of the final product can be replaced by the free storage during the production.

There are some approaches to making formulation of this problem closer to real life. Either by assignment of the due date, earliness cost, and tardiness cost to all task (Baptiste, Flamini, & Sourd To appear in 2008), or by introduction of buffers with limited capacity that are used during the production (Brucker *et al.* 2006).

The approach with limited buffers is also used in the formulation of the lacquer production scheduling (Kelbel & Hanzálek 2006) where each job needs a limited buffer (mixing vessel) during the whole time of its execution.

In future, we would like to focus on the formulation and solution of the job shop problems with earliness and tardiness costs and with generic limited buffers.

Acknowledgement

The work described in this paper was supported by the Czech Ministry of Education under Project 1M0567. Also, we would like to thank the anonymous reviewers for useful comments.

References

Baker, K. R., and Scudder, G. D. 1990. Sequencing with earliness and tardiness penalties: A review. *Operations Research* 38(1):22–36.

Baptiste, P.; Flamini, M.; and Sourd, F. To appear in 2008. Lagrangian bounds for just-in-time job-shop scheduling. *Computers & Operations Research* 35(3):906–915.

Barták, R. 1999. Constraint programming – what is behind? In *Proc. of CPDC99 Workshop*.

Beck, J. C., and Perron, L. 2000. Discrepancy-bounded depth first search. In *Second International Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems (CP-AI-OR'00)*.

Beck, J. C., and Refalo, P. 2001. A hybrid approach to scheduling with earliness and tardiness costs. In *Third International Workshop on Integration of AI and OR Techniques (CP-AI-OR'01)*.

Beck, J. C., and Refalo, P. 2002. Combining local search and linear programming to solve earliness/tardiness scheduling problems. In *Fourth International Workshop on Integration of AI and OR Techniques (CP-AI-OR'02)*.

Beck, J. C., and Refalo, P. 2003. A hybrid approach to scheduling with earliness and tardiness costs. *Annals of Operations Research* 118(1–4):49–71.

Brucker, P.; Heitmann, S.; Hurink, J.; and Nieberg, T. 2006. Job-shop scheduling with limited capacity buffers. *OR Spectrum* 28(2):151–176.

Carlier, J., and Pinson, E. 1990. A practical use of jackson's pre-emptive schedule for solving the job-shop problem. *Annals of Operations Research* 26:269–287.

Danna, E., and Perron, L. 2003. Structured vs. unstructured large neighborhood search: A case study on job-shop

size	10 × 10						15 × 10						20 × 10					
lf	MIP		uLNS		CDI		MIP		uLNS		CDI		MIP		uLNS		CDI	
	F	P	F	P	F	P	F	P	F	P	F	P	F	P	F	P	F	P
1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1.3	0	0	0	0	0	0	1	1	2	2	5	0	0	0	0	0	0	0
1.5	7	7	9	9	10	4	5	5	9	9	12	3	3	3	4	4	5	3

Table 2: Number of optimal solutions (F)ound and (P)roven within 600 s

size	10 × 10						15 × 10						20 × 10					
lf	MIP		uLNS		CDI		MIP		uLNS		CDI		MIP		uLNS		CDI	
	B	U	B	U	B	U	B	U	B	U	B	U	B	U	B	U	B	U
1.0	0	0	5	5	15	15	0	0	0	0	20	20	0	0	0	0	20	20
1.3	0	0	2	2	18	18	1	0	2	0	20	18	1	1	1	1	18	18
1.5	8	0	12	0	20	8	5	0	14	3	16	6	3	0	7	4	16	12

Table 3: Number of (B)est and (U)niquely best solutions found within 600 s

scheduling problems with earliness and tardiness costs. In *Ninth International Conference on Principles and Practice of Constraint Programming*, 817–821.

Danna, E.; Rothberg, E.; and Le Pape, C. 2005. Exploring relaxation induced neighborhoods to improve MIP solution. *Mathematical Programming* 102(1):71–90.

El Sakkout, H., and Wallace, M. 2000. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints* 5(4):359–388.

Harvey, W. D., and Ginsberg, M. L. 1995. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 607–615.

ILOG S.A. 2002. *ILOG OPL Studio 3.6 Language Manual.*

ILOG S.A. 2005. *ILOG Cplex 9.1 User's Manual.*

Kelbel, J., and Hanzálek, Z. 2006. A case study on earliness/tardiness scheduling by constraint programming. In *Proceedings of the CP 2006 Doctoral Programme*, 108–113.

Taillard, E. 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research* 64:278–285.

Van Hentenryck, P.; Perron, L.; and Puget, J.-F. 2000. Search and strategies in OPL. *ACM Transactions on Computational Logic* 1(2):285–320.

Single-machine Scheduling with Tool Changes: A Constraint-based Approach

András Kovács
Computer and Automation Research Institute
Hungarian Academy of Sciences
akovacs@sztaki.hu

J. Christopher Beck
Dept. of Mechanical & Industrial Engineering
University of Toronto
jcb@mie.utoronto.ca

Abstract

The paper addresses the scheduling of a single machine with tool changes in order to minimize total completion time. A constraint-based model is proposed that makes use of global constraints and also incorporates various dominance rules. With these techniques, our constraint-based approach outperforms previous exact solution methods.

Introduction

This paper addresses the problem of scheduling a single machine with tool changes, in order to minimize the total completion time of the activities. The regular replacement of the tool is necessary due to wear, which results in a limited, deterministic tool life. We note that this problem is mathematically equivalent to scheduling with periodic preventive maintenance, where there is an upper bound on the continuous running time of the machine. After that, a fixed-duration maintenance activity has to be performed.

Our main intention is to demonstrate the applicability of constraint programming (CP) to an optimization problem that requires complex reasoning with constraints on sum-type expressions, a field were CP is generally thought to be in handicap. We show that indeed, when appropriate global constraints are available to deal with such expressions, CP outperforms other exact optimization techniques. In particular, we would like to illustrate the efficiency of the global COMPLETION constraint (Kovács & Beck 2007), which has been proposed recently for propagating the total weighted completion time of activities on a single unary resource.

For this purpose, we define a constraint model of the scheduling problem. The model makes use of global constraints, and also incorporates various dominance properties described as constraints. A simple branch and bound search is used for solving the problem. We show in computational experiments that the proposed approach can outperform all previous exact optimization methods known for this problem.

The paper is organized as follows. After reviewing the related literature, we give a formal definition of the problem and outline some of its basic characteristics. Then, we propose a constraint-based model of the problem. The algorithms used for propagating the global constraints that are crucial for the performance of our solver are presented. Afterwards, the branch and bound search procedure used is introduced. Finally, experimental results are presented and conclusions are drawn.

Related Work

The problem studied in this paper has been introduced independently in the periodic maintenance context by Qi, Chen, & Tu (1999) and in the tool changes context by Akturk, Ghosh, & Gunes (2003). Its practical relevance is underlined in (Gray, Seidmann, & Stecke 1993), where it is pointed out that in many industries tool change induced by wear is ten times more frequent than change due to the different requirements of subsequent activities. Also, in some industries, e.g. in metal working, tool change times can dominate actual processing times (Tang & Denardo 1988).

Akturk, Ghosh, & Gunes (2003) proposed a mixed-integer programming (MIP) approach and compared the performance of various heuristics on this problem. The basic properties of the scheduling problem have been analyzed and the performance of the Shortest Processing Time (SPT) schedules evaluated in (Akturk, Ghosh, & Gunes 2004). Three different heuristics have been analyzed and a branch and bound algorithm proposed by Qi, Chen, & Tu (1999). The performance of four different MIP models have been compared in (Chen 2006a).

The same problem has been considered with different objective criteria, including makespan (Chen 2007b; Ji, He, & Cheng 2007), maximum tardiness (Liao & Chen 2003), and total tardiness (Chen 2007a). In (Akturk, Ghosh, & Kayan 2007), the model is extended to controllable activity durations, where there are several execution modes available for each activity to balance between manufacturing speed and tool wear. The basic model with several tool types has been investigated by Karakayalı & Azizoğlu (2006). A slightly different problem, in which maintenance periods are strict, i.e. the machine has to wait idle if activities complete earlier than the end of the period, has been investigated in (Chen 2006b).

A brief introduction to constraint-based scheduling is given in (Barták 2003), while an in-depth presentation of the modeling and solution techniques can be found in (Baptiste, Le Pape, & Nuijten 2001).

Problem Definition and Notation

There are n non-preemptive activities A_i to be scheduled on a single machine. Activities are characterized by their durations p_i, and are available from time 0. Processing the activities requires a type of tool that is available in an unlimited number, but has a limited tool life, TL. Worn tools can be replaced with a new one, but only without interrupting activities. This change requires TC time. It is assumed that $\forall i\ p_i \leq TL$, because otherwise the problem would have no solution. The objective is to determine the start times S_i of the activities and start times t_j of tool changes such that the total completion time of the activities is minimal.

Constraint programming uses inference during search on the current domains of the variables. The minimum and maximum values in the current domain of a variable X will be denoted by $\check{X}$ and $\hat{X}$, respectively. Hence, $\check{S}_i$ will stand for the earliest start time of activity A_i, and $\hat{C}_i$ for its latest finish time.

The above parameters and the additional notation used in the paper is summarized in Fig. 1. We assume that all data are integral. A sample schedule is presented in Fig. 2.

n	- Number of activities
p_i	- Duration of activity A_i
p_{max}	- Maximum duration of activities A_i
TL	- Tool life
TC	- Tool change time
S_i	- Start time of activity A_i
C_i	- End (completion) time of activity A_i
t_j	- (Start) time of the jth tool change
a_j	- Number of activities processed after the jth tool change
b_j	- Number of activities processed before the jth tool change
$\check{X}$	- Minimum value in the domain of variable X
$\hat{X}$	- Maximum value in the domain of variable X

Figure 1: Notation

Basic Properties

The single-machine scheduling problem with tool changes, denoted as $1|tool-changes|\sum_i C_i$, has been proven to be NP-hard in the strong sense in (Akturk, Ghosh, & Gunes 2004). The same paper and (Qi, Chen, & Tu 1999) investigated properties of optimal solutions. Below we outline these properties, in conjunction with a symmetry breaking rule that can also be exploited to increase the efficiency of solution algorithms.

Property 1 (No-wait schedule) Activities must be scheduled without any waiting time between them, apart from the tool change times.

Property 2 (SPT within tool) Activities executed with the same tool must be sequenced in the SPT order.

Property 3 (Tool utilization) The total duration of activities processed with the jth tool is at least $TL - p_j^{minafter} + 1$, where $p_j^{minafter}$ is the minimal duration of activities processed with tools $j' > j$.

Consequence Every tool, except for the last one, is utilized during at least $U_{min} = TL - p_{max} + 1$ time, where p_{max} is the largest activity duration. Hence, the number of tools required is at most $\lceil \sum_{i=1}^{n} p_i / U_{min} \rceil$.

Property 4 (Activities per tool) The number of activities processed using the jth tool is a non-increasing function of j.

Property 5 (Symmetry breaking) There exists an optimal schedule in which for any two activities A_i and A_j such that $p_i = p_j$ and $i < j$, A_i precedes A_j.

Modeling the Problem

In our constraint model we apply a so-called *machine time* representation, which considers only the active periods of the machine. It exploits that the optimal solution is a no-wait schedule (see Property 1), and contracts each tool change into a single point in time, as shown in Fig. 3. Then, a solution corresponds to a sequencing of the activities, with the last activity ending at $\sum_i p_i$, and instantaneous tool changes between them.

The objective value of a schedule in the machine time representation takes the form

$$\sum_{i=1}^{n} C_i + TC \sum_{j=1}^{m} a_j.$$

Technically it will be easier to work with b_j than with a_j, hence, we rewrite the objective function to the equivalent form

$$\sum_{i=1}^{n} C_i + TC \sum_{j=1}^{m} (n - b_j).$$

We decompose this function to $K_1 = \sum_{i=1}^{n} C_i$ and $K_2 = TC \sum_{j=1}^{m} (n - b_j)$. Note that K_1 corresponds to the total completion time without tool changes, while K_2 represents the effect of introducing tool changes.

The variables in the model are the start times S_i of the activities, the times t_j of the tool changes, and the number of activities processed before the jth tool change, b_j. The two cost components K_1 and K_2 are also handled as model variables. For the sake of brevity,

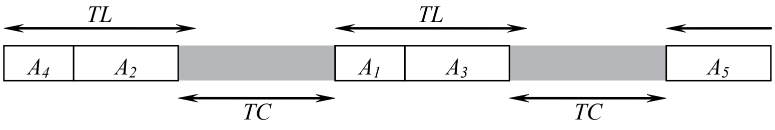


Figure 2: A sample schedule. Wall clock time representation.

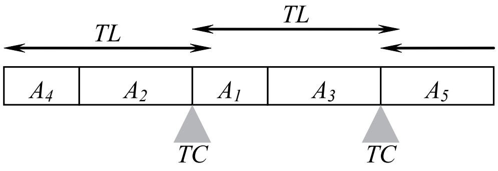


Figure 3: Machine time representation of the sample schedule.

we also use $C_i = S_i + p_i$ to denote the end time of activity A_i.

Then, the problem consists of minimizing $K_1 + K_2$ subject to

(c1) Time window constraints, stating $\forall i: \ S_i \geq 0$ and $C_i \leq \sum_i p_i$;

(c2) Resource capacity constraint: at most one activity can be processed at any point in time;

(c3) Activities are not interrupted by tool changes: $\forall i, j$: $C_i \leq t_j \ \vee \ S_i \geq t_j$;

(c4) Limited tool life: $\forall j: \ t_{j+1} - t_j \leq TL$;

(c5) Property 3 holds: $\forall j: \ t_{j+1} - t_j \geq TL - p_{max} + 1$;

(c6) Property 4 holds: $\forall j: \ b_j - b_{j-1} \geq b_{j+1} - b_j$;

(c7) Property 5 holds: $\forall i_i, i_2$ such that $i_1 < i_2$ and $p_{i1} = p_{i2}$: $C_{i1} \leq S_{i2}$;

(c8) The total completion time of activities A_i is K_1;

(c9) The number of activities that end before t_j is b_j;

(c10) $K_2 = TC \sum_{j=1}^{m} (n - b_j)$.

Note that while constraints c1-c4 and c8-c10 are fundamental elements of our model, c5-c7 incorporate dominance rules to facilitate stronger pruning of the search space. All the ten constraint can be expressed by languages of common constraint solvers. However, significant improvement in performance can be achieved by applying dedicated global constraints for propagating c8 and c9. We discuss those global constraints in detail in the next section.

Propagation Algorithms for Global Constraints

Below, both for c8 and c9, we first present how the constraint can be expressed in typical constraint languages. Then, we introduce a dedicated global constraint and a corresponding propagation algorithm for either of them, in order to strengthen pruning.

Total Completion Time

The typical way of expressing the total completion time of a set of activities in constraint-based scheduling is posting a sum constraint on their end times: $K = \sum C_i$. However, the sum constraint, ignoring the fact that the activities require the same unary resource, assumes that all of them can start at their earliest start times. This leads to very loose initial lower bounds on K; in the present application $\check{K} = \sum_i p_i$.[1]

In order to achieve tight lower bounds on K and strong back propagation to the start time variables S_i, the COMPLETION constraint has been introduced in (Kovács & Beck 2007) for the total weighted completion time of activities on a unary capacity resource. Formally, it is defined as

$$\text{COMPLETION}([S_1, ..., S_n], [p_1, ..., p_n], [w_1, ..., w_n], K)$$

and enforces $K = \sum_i w_i(S_i + p_i)$. Checking generalized bounds consistency on the constraint requires solving $1|r_i, d_i| \sum w_i C_i$, a single machine scheduling problem with release times and deadlines and upper bound on the total weighted completion time. This problem is NP-hard, hence, cannot be solved efficiently each time the COMPLETION constraint has to be propagated. Instead, our propagation algorithm filters domains with respect to the following *relaxation* of the above problem.

The *preemptive mean busy time* relaxation (Goemans *et al.* 2002), denoted by $1|r_i, pmtn| \sum w_i M_i$, involves scheduling preemptive activities on a single machine with release times respected, but deadlines disregarded. It minimizes the total weighted mean busy times M_i of the activities, where M_i is the average point in time at which the machine is busy processing A_i. This is easily calculated by finding the mean of each time point at which activity A_i is executed. This relaxed problem can be solved to optimality in $O(n \log n)$ time.

[1] The lower bound is a little tighter if symmetry breaking constraints (c7) are present to increase the earliest start times of some activities.

The COMPLETION constraint filters the domains of the start time variables by computing the cost of the optimal preemptive mean-busy time relaxation *for each activity* A_i and *each possible start time* t of activity A_i, with the added constraint that activity A_i must start at time t. If the cost of the relaxed solution is greater than the current upper bound, then t is removed from the domain of S_i. The naive computation of all these relaxed schedules is likely to be too expensive, computationally. The main contribution of (Kovács & Beck 2007) is showing that for each activity it is sufficient to compute relaxed solutions for a limited number of different values of t, and that subsequent relaxed solutions can be computed iteratively by a permutation of the activity fragments in previous solutions. For a detailed presentation of this algorithm and the COMPLETION constraint, in general, readers are referred to the above paper.

Number of Activities before a Tool Change

Constraint c8 describes a complex global property of the schedule. Standard CP languages make it possible to express this property with the help of binary logical variables indicating whether a given activity ends before a point in time, i.e.

$$y_{i,j} = \begin{cases} 1 & \text{if } C_i \le t_j \\ 0 & \text{otherwise.} \end{cases}$$

Then, b_j can be computed as $b_j = \sum_i y_{i,j}$. This representation would be rather inefficient, but implementing a global constraint for this purpose is rather straightforward.

The NBEFORE global constraint states that given activities A_i that have to be executed on the same unary resource, the number of activities that can be completed before time t_j is exactly b_j:

$$\text{NBEFORE}([S_1, ..., S_n], t_j, b_j)$$

The propagation algorithm for this global constraint is presented in Fig. 4. It first determines the set of activities M that *must be* executed before t_j, and the set of activities P that are *possibly* executed before t_j. Computing the minimal (maximal) number of activities scheduled before t_j is performed by sorting P by non-decreasing duration, and then selecting the activities that have the highest (lowest) durations. The algorithm completes by updating $\check{b}_j$, $\hat{b}_j$, and $\check{t}_j$. The time complexity of the propagator is $O(n \log n)$, which is the time needed for sorting P.

We note that it is straightforward to extend this algorithm with propagation from m_j and t_j to S_i, and also to $\hat{t}_j$. This extension has been implemented, but did not achieve additional pruning, and therefore it has been later omitted.

A Branch and Bound Search

We apply a branch and bound search that exploits the dominance properties identified for the problem. It constructs a schedule chronologically, by fixing the start times of activities and the times of tool changes. In each node it selects, according to the SPT rule, the minimal duration unscheduled activity A^* that can be scheduled next. The algorithm first checks if one of the following dominance rules can be applied at this phase of the search.

- If the remaining activities can all be scheduled without any tool changes, then A^* must be scheduled immediately, because all the unscheduled activities must be scheduled according to the SPT rule. See Property 2 and lines 4-5 of the algorithm.
- If A^* cannot be performed before the next tool change, then no unscheduled activities can be performed before the next tool change, since none of them have shorter durations than A^*. Therefore the next tool change must be performed immediately. See Property 1 and lines 6-7 of the algorithm.

If one of the dominance rules can be applied, then the algorithm adds the inferred constraint, which may trigger further propagation, and then reselects A^* w.r.t. the new variable domains. Otherwise, it creates two children of the current search node, according to whether

- A^* is scheduled immediately and the next tool change is performed after (but not necessarily immediately after) A^*; or
- A^* is scheduled after the next tool change.

In the latter case, it also adds the constraint that another activity must be scheduled before the next tool change. Hence, the next tool change must be performed after C_{min}, which is the lowest among the end times of unscheduled activities (see line 9). Note that C_{min} exists because if there is an unscheduled activity (A^*), then there are at least two unscheduled activities.

Also observe that the initial solution found by this branch and bound algorithm is the SPT schedule.

Experimental Result

We ran computational experiments to evaluate the performance of the proposed CP approach from several aspects. We addressed understanding how the COMPLETION and NBEFORE global constraints improve the performance of our model compared to models using only tools of standard CP solvers. We also measured how problem characteristics influence the performance of our approach, and finally, we compared it to previous exact solution methods.

All models and algorithms have been implemented in Ilog Solver and Scheduler version 6.1. The experiments were run on a 2.53 GHz Pentium IV computer with 760 MB of RAM.

Two different problem sets were used for the experiments. The first set was generated as instances in (Qi, Chen, & Tu 1999), the second as in (Akturk, Ghosh, & Gunes 2003). Qi, Chen, & Tu (1999) took activity durations randomly from the interval $[1, 30]$ and fixed

```
PROCEDURE Propagate()
  $M = \{A_i \mid \hat{S}_i < \check{t}_j\}$
  $P = \{A_i \mid \check{C}_i \leq \hat{t}_j\} \setminus M$
  Sort $P$ by non-decreasing duration
  $k_{min}$ = min number of activities in $P$ with total duration $\geq \check{t}_j - \sum_{A_i \in M} p_i$
  $k_{max}$ = max number of activities in $P$ with total duration $\leq \hat{t}_j - \sum_{A_i \in M} p_i$
  $\check{b}_j = |M| + k_{min}$
  $\hat{b}_j = |M| + k_{max}$
  $\check{t}_j$ = $\sum_{A_i \in M} p_i$ + total duration of the $k_{min}$ shortest activities in $|P|$
```

Figure 4: Algorithm for propagating the NBEFORE constraint.

```
WHILE there are unscheduled activities
  $A^*$ = Unscheduled activity with min $\check{S}_{A^*}$, min $p_{A^*}$
  $T$ = Earliest tool change time with $\hat{T} > \check{S}_{A^*}$
  IF there is no such $T$
    ADD $S_{A^*} = \check{S}_{A^*}$ (Property 2)
  ELSE IF $\hat{T} < \check{C}_{A^*}$
    ADD $T = \check{S}_{A^*}$ (Property 1)
  ELSE
    $C_{min}$ = min $\check{C}_i$ of unscheduled activities $A_i \neq A^*$
      BRANCH:      - $S_A = \check{S}_A$ and $C_A \leq T$
                   - $S_A \geq T$ and $T \geq C_{min}$
```

Figure 5: Pseudo-code of the search algorithm.

the value of TC to 10. The number of activities n has been varied between 15 and 40 in increments of 5, while values of the tool life TL have been taken from $\{50, 60, 70, 80\}$. We generated ten instances with each combination of n and TL, which resulted in 240 problem instances. The time limit for these problems was set to one hour.

In (Akturk, Ghosh, & Gunes 2003), in order to obtain instances with different characteristics, four parameters of the generator were varied, each having a low (L) and a high (H) value. These parameters were the mean and the range of the durations (MD and RD), the tool life (TL), and the tool change time (TC). Generating ten 20-activity instances with each combination of the parameters resulted in $2^4 \cdot 10 = 160$ instances. Since this set contains harder instances, we set the time limit to two hours.

We did not perform comparisons with the MIP models proposed in (Chen 2006a), because that paper presents experimental results only on very easy instances containing few (in most cases only one) tool changes over the scheduling horizon.

Results on Qi's Instances and Comparison to Naive Models

We compared the performance of four different CP models of the problem that represent the two cost components K_1 and K_2 in different ways. K_1 was expressed either by a sum constraint (*Sum*)or by the COMPLETION constraint (*COMPL*), while K_2 was described using binary variables (*Bin*) or the NBEFORE constraint (*NBEF*). Note that the *COMPL/NBEF* is the model proposed in this paper.

The achieved results are displayed in Table 1. Each row contains cumulative results for ten instances with a given value of n and TL. For each of the models, column *Opt* shows the number of instances for which the optimal solution has been found and optimality has been proven, column *Nodes* contains the average number of search nodes, and *Time* the average search time in seconds. *Nodes* and *Time* also contain the effort needed for proving optimality.

The results show that the proposed approach, *COMPL/NBEF* solves instances with up to 30-35 activities to optimality. It outperforms the alternative CP representations that do not benefit from the pruning strength of the COMPLETION and NBEFORE constraints. Instances with a short tool life and hence, many tool changes are more challenging. This is due to the poorer performance of the SPT heuristic, and higher importance of the bin packing aspect of the problem. In contrast, Qi, Chen, & Tu (1999) report that the average solution time of 20-activity instances with their branch and bound approach was in the range of [55.94, 3.57] seconds, depending on the value of TL, and their algorithm could not cope with larger problems.

n	TL	Sum/Bin			COMPL/Bin			Sum/NBEF			COMPL/NBEF		
		Opt	Nodes	Time	Opt	Nodes	Time	Opt	Nodes	Time	Opt	Nodes	Time
15	50	10	36278	10.8	10	877	0.0	10	31134	5.4	10	49	0.0
	60	10	55477	13.6	10	1018	0.2	10	49975	7.7	10	76	0.0
	70	10	18275	3.1	10	358	0.0	10	14357	1.5	10	17	0.0
	80	10	19748	2.9	10	303	0.0	10	15502	1.4	10	19	0.0
20	50	6	5365305	2605.5	10	42853	35.1	8	6579567	1685.3	10	7183	3.7
	60	7	5365603	1778.5	10	19092	16.2	7	7511826	1436.0	10	133	0.0
	70	9	2544734	735.1	10	8051	7.1	9	3119249	558.0	10	84	0.0
	80	10	910496	241.8	10	1957	1.4	10	762404	127.8	10	46	0.0
25	50	0	6282502	3600.0	10	639147	727.3	0	11727713	3600.0	10	99239	78.0
	60	0	9132083	3600.0	10	91385	126.4	0	15404729	3600.0	10	1126	0.4
	70	1	10815570	3587.7	10	83095	104.2	2	16222223	3327.3	10	979	0.2
	80	1	11484097	3358.2	10	91029	122.1	1	16808958	3287.7	10	1082	0.6
30	50	-	-	-	3	2581475	3229.5	-	-	-	9	230088	452.5
	60	-	-	-	4	2093233	2804.0	-	-	-	10	55374	46.9
	70	-	-	-	8	961460	1640.2	-	-	-	10	7877	6.6
	80	-	-	-	10	318435	560.9	-	-	-	10	1721	1.1
35	50	-	-	-	0	3108739	3600.0	-	-	-	7	1724651	2002.6
	60	-	-	-	0	3193284	3600.0	-	-	-	9	355709	449.5
	70	-	-	-	0	2858550	3600.0	-	-	-	10	160239	166.9
	80	-	-	-	2	2000949	3162.0	-	-	-	10	8121	8.9
40	50	-	-	-	-	-	-	-	-	-	1	2371440	3297.7
	60	-	-	-	-	-	-	-	-	-	6	1088871	1597.6
	70	-	-	-	-	-	-	-	-	-	10	279844	393.5
	80	-	-	-	-	-	-	-	-	-	10	85854	143.3

Table 1: Experimental results on instances from (Qi, Chen, & Tu 1999): number of instances where optimality has been proven (Opt), average number of search nodes (Nodes), and average solution time in seconds (Time), for four different CP models. The models use binary variables (*Bin*) or the *NBEFORE* constraint, and a *Sum* or a *COMPLETION* constraint to express the objective function. Dash '-' means that none of the instances with the given n could be solved to optimality.

Results on Akturk's Instances and Effect of Problem Characteristics

Experimental results on the instances from (Akturk, Ghosh, & Gunes 2003) are presented in Table 2. The results on the l.h.s. have been achieved by a naive model with sum back propagation instead of the COMPLETION constraint, the results on the r.h.s. by the complete CP model.

Each row displays data belonging to a given choice of parameters MD, RD, TL, and TC, as shown in the leftmost columns. While the COMPLETION model managed to solve all instances to optimality and also proved optimality, the sum model missed finding the optimum for 2 instances and proving optimality in 5 cases. The COMPLETION model was 10 times faster on average than the sum model.

These results confirm that short tool life implies many tool changes and renders problems more complicated for our model. Low mean duration makes things easier, which is probably due to the higher number of symmetric activities, since these activities can be ordered a priori. Although a low range of durations has a similar effect, it also has a negative impact on the performance of the SPT heuristic, among which the latter seems to be the stronger.

Compared to the MIP approach presented in (Akturk, Ghosh, & Gunes 2003) our CP model solves more instances, and does this more quickly: the MIP model achieved an average solution time of 1904 seconds, it was not able to solve all instances, and for the 15% of the instances it found worse solutions than one of the heuristics.

Conclusion

A constraint-based approach has been presented to single machine scheduling with tool changes. The proposed model outperforms previous exact optimization methods known for this problem. This result is significant especially because the problem requires complex reasoning with sum-type formulas, which does not belong to the traditional strengths of constraint programming. This was made possible by two algorithmic techniques: global constraints and dominance rules. Specifically, we applied the recently introduced COMPLETION constraint to propagate total completion time, and defined a new global constraint, NBEFORE, to compute the number of activities that complete before a given point in time. Furthermore, we could formulate the known dominance properties as constraints in the model.

The introduced model can be easily extended with constraints on the number of tools and with weighted activities. The machine-time representation is appli-

MD	*RD*	*TL*	*TC*	NBEF/Sum				NBEF/COMPL			
				Opt	MRE	Nodes	Time	Opt	MRE	Nodes	Time
L	L	L	L	10	0	1891018	529.9	10	0	38128	23.3
L	L	L	H	10	0	968087	205.9	10	0	102237	52.1
L	L	H	L	10	0	79344	11.9	10	0	237	0.1
L	L	H	H	10	0	12269	1.6	10	0	73	0.0
L	H	L	L	10	0	667659	171.8	10	0	3692	2.3
L	H	L	H	10	0	127866	23.7	10	0	78955	25.7
L	H	H	L	10	0	78775	13.2	10	0	27	0.0
L	H	H	H	10	0	6664	0.7	10	0	29	0.0
H	L	L	L	7	1.71	16430139	3548.8	10	0	1614494	596.4
H	L	L	H	10	0	5606737	1018.0	10	0	47902	25.1
H	L	H	L	10	0	2170750	357.9	10	0	895	0.3
H	L	H	H	10	0	222435	40.6	10	0	9023	3.6
H	H	L	L	8	0	6020041	2102.8	10	0	81249	43.9
H	H	L	H	10	0	186735	35.7	10	0	23214	11.3
H	H	H	L	10	0	86856	12.5	10	0	20	0.0
H	H	H	H	10	0	154639	19.2	10	0	1648	0.8

Table 2: Experimental results on instances from (Akturk, Ghosh, & Gunes 2003), for models using sum and COMPLETION back propagation: number of instances where optimality has been proven (Opt), mean relative error in percents (MRE), average number of search nodes (Nodes), and average solution time in seconds (Time).

cable to solving the same problem with other regular optimization criteria, such as minimizing makespan, or maximum or total tardiness. However, it seems to be impractical to apply this model to multiple-machine problems, because the time scales would differ machine by machine.

Acknowledgments The authors are grateful to the anonymous reviewers for their helpful comments. A. Kovács was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and by the NKFP grant 2/010/2004.

References

Akturk, M. S.; Ghosh, J. B.; and Gunes, E. D. 2003. Scheduling with tool changes to minimize total completion time: A study of heuristics and their performance. *Naval Research Logistics* 50:15–30.

Akturk, M. S.; Ghosh, J. B.; and Gunes, E. D. 2004. Scheduling with tool changes to minimize total completion time: Basic results and SPT performance. *European Journal of Operational Research* 157:784–790.

Akturk, M. S.; Ghosh, J. B.; and Kayan, R. K. 2007. Scheduling with tool changes to minimize total completion time under controllable machining conditions. *Computers and Operations Research* 34:2130–2146.

Baptiste, P.; Le Pape, C.; and Nuijten, W. 2001. *Constraint-based Scheduling.* Kluwer Academic Publishers.

Barták, R. 2003. Constraint-based scheduling: An introduction for newcomers. In *Intelligent Manufacturing Systems 2003*, 69–74.

Chen, J.-S. 2006a. Single-machine scheduling with flexible and periodic maintenance. *Journal of the Operational Research Society* 57:703–710.

Chen, W. J. 2006b. Minimizing total flow time in the single-machine scheduling problem with periodic maintenance. *Journal of the Operational Research Society* 57:410–415.

Chen, J.-S. 2007a. Optimization models for the tool change scheduling problem. *Omega* (to appear).

Chen, J.-S. 2007b. Scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan. *European Journal of Operational Research* (to appear).

Goemans, M. X.; Queyranne, M.; Schulz, A. S.; Skutella, M.; and Wang., Y. 2002. Single machine scheduling with release dates. *SIAM Journal on Discrete Mathematics* 15(2):165–192.

Gray, E.; Seidmann, A.; and Stecke, K. E. 1993. A synthesis of decision models for tool management in automated manufacturing. *Management Science* 39:549–567.

Ji, M.; He, Y.; and Cheng, T. C. E. 2007. Single-machine scheduling with periodic maintenance to minimize makespan. *Computers & Operations Research* 34:1764–1770.

Karakayalı, I., and Azizoğlu, M. 2006. Minimizing total flow time on a single flexible machine. *International Journal of Flexible Manufacturing Systems* 18:55–73.

Kovács, A., and Beck, J. C. 2007. A global constraint for total weighted completion time. In *Proceedings of CPAIOR'07, 4th Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Com-*

binatorial Optimization Problems (LNCS 4510), 112–126.

Liao, C. J., and Chen, W. J. 2003. Single-machine scheduling with periodic maintenance and nonresumable jobs. *Computers & Operations Research* 30:1335–1347.

Qi, X.; Chen, T.; and Tu, F. 1999. Scheduling the maintenance on a single machine. *Journal of the Operational Research Society* 50:1071–1078.

Tang, C. S., and Denardo, E. V. 1988. Models arising from a flexible manufacturing machine, Part I: Minimization of the number of tool switches. *Operations Research* 36:767–777.

Comprehensive approach to University Timetabling Problem

Wojciech Legierski, Łukasz Domagała

Silesian University of Technology, Institute of Automatic Control, 16 Akademicka str., 44-100 Gliwice, Poland
Wojciech.Legierski@polsl.pl, Lukasz.Domagala@student.polsl.pl

Abstract

The paper proposes a comprehensive approach to University Timetabling Problem, presents a constraint-based approach to automating solving and describes a system that allows concurrent access by multiple users. The timetabling needs to take into account a variety of complex constraints and uses special-purpose search strategies. Local search incorporated into the constraint programming is used to further optimize the timetable after the satisfactory solution has been found. Paper based on experience gained during implementation of the system at the Silesian University of Technology, assuming coordinating the work of above 100 people constructing the timetable and above 1000 teachers who can have influence on timetabling process. One of the original issues used in real system, presented in the paper, is multi-user access to the timetabling database giving possibility of offline work, solver extended by week definitions and dynamic resource assignment.

Introduction

Timetabling is regarded as a hard scheduling problem, where the most complicated issue is the changing of requirements along with the institution for which the timetable is produced. Automated timetabling can be traced back to the 1960s [Wer86]. Some trials of comprehensively approaching the timetabling problem are presented in the timetabling research center lead by prof. Burke [PB04] and in several PhD thesis [M05],[Rud01],[Mar02]. There are works connected with general data formulation , metaheuristic approaches, and user interfaces for timetabling. The paper presents a proposition of a comprehensive approach to the real-world problem at Silesian University of Technology. This paper presents the methods used for automated timetabling, data description and user interaction underlining connection of different idea to built whole timetabling system.

Problems description

A number of timetabling problems have been discussed in the literature [Sch95]. Based on the detailed classification proposed by Reise and Oliver [RL01], the presented problem consists of a mixture of following categories:

Class-Teacher Timetabling (CTT) – the problem amounts to allocating a timeslot to each course provided for a class of students that has a common programme,
Room Assignment (RA) - each course has to be placed in a suitable room (or rooms), with a sufficient number of seats and equipment needed by this course.
Course Timetabling (CT) - the problem assumes that students can choose courses and need not belong to some classes.
Staff Allocation (SA) - the problem consists of assigning teachers to different courses, taking into account their preferences. The problem assumes that one course can be conducted by several teachers.

Till now Examination Timetabling (ET) was not required, but is planned to be added in future.

Comprehensive approach to University Timetabling Problem (UTP), besides taking into account different timetabling problems, also assumes following tasks:

- formulating timetable data requires a lot of flexibility,
- automated methods should be available for sub-problems and should be able to take into account many soft and hard constraints,
- timetabling can be conducted by many users, simultaneously, which requires assistance in manual timetabling and quick availability to different resources' plans.

Constraints

Timetable of UTP has to fulfill the following constraints, which can be expressed as hard or soft:

- resources assigned to a course (classes, teachers, rooms, students) have time of unavailability and undesirability,
- courses with the same resource cannot overlap,
- some courses must be run simultaneously or in defined order,
- some resources can be constrained not to have courses in all days and more than some number during a day,
- no gaps constraint between courses for the same resource or gaps between some specific courses can be strictly defined,

- the number of courses per day should be roughly equal for defined resource - p,
- courses should start from early morning hours.

Data representation

Although UTP data is gathered in relational database for multi user access, data for the solver is saved as a XML file, which also expresses sub-problems for the solver. The main advantage of using XML file is the ability of defining relations between courses, resources and constraints in a very flexible way. The flexibility of UTP features:

- defining arbitrarily resources (classes, teachers, rooms, students)
- allowing assignment of some different resources to one course,
- assigning resources can be treated as disjunction of some resources, where also a number of chosen resources can be defined,
- constraints can be imposed on every resource and every course,

Additionally in UTP we are supposed to produce a plan which is coherent during a certain time-span (it would be for example one semester), with courses taking place cyclically with some period (most often one-week period). But frequently we face a situation where some courses do not fit into the period mentioned above, for example some of them should appear only in odd weeks or only in even weeks and thus have a two week period. Seeking solution to this problem we introduced the idea of “week definition”. Different week definitions can be defined in the timetable, together with the information, which of them have common weeks and the courses assigned to them.

Multi user access

The UTP requires taking into account that there are many timetable designers, who are engaged in timetabling process. The teachers and students are asked to submit information about their choices as well as time preferences. The appropriate management of the user interaction is solved by introducing 3 levels of the rights assigned to each user and connected with set of resources like groups, teachers, students and rooms:

- user can be administrator of the resource,
- user can be planner of the resource,
- user can only use resource for courses.

Additionally each resource and courses have user which is call “owner”. Owners and administrators can block resources to restrict changing them.

Manual timetabling assistance

Timetable designers often do not want to introduce all the constraints and trust the computer in putting courses in the best places. Manual timetabling assistance with constraint explanation seems to be a very important step in making timetable system useful. The assistance requires very quick access to a lot of data and relations between them to provide a satisfactory interface. Therefore after dragging the course, colors of unavailable timeslots change to color defining what sort of constraints will be violated. For example overlapping of rooms courses has gray color and undesirable hours of a teacher leaded the course has yellow color.

Structure of the system

The proposed solution for comprehensive approach to UTP requires a usage of different languages and technologies for different features. Therefore the proposed system consists of 4 parts as presented in Figure.1. The system was firstly presented by the author in [LW03]. Presented system was extended mainly by multi-user access.

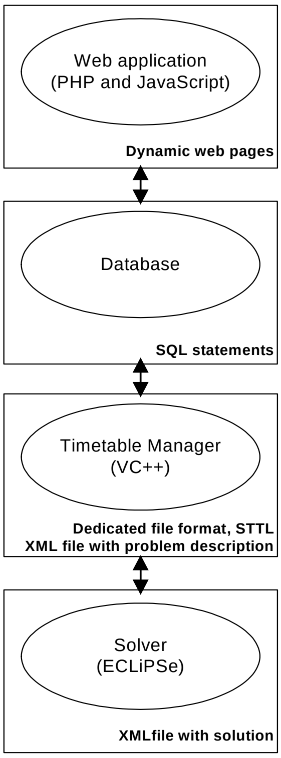


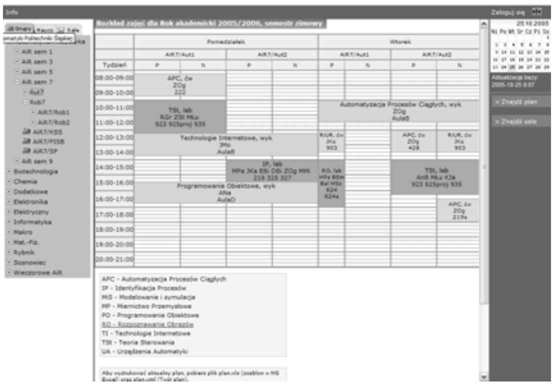
Figure 1, Diagram of four parts of the system, their dependencies and their output data format.

Web application

HTML seems to be an obvious solution for presenting results of the timetabling process in the Internet, but it provides only static pages which are not sufficient for

SAT. JavaScript improves the user interface and provides the capability to create dynamic pages.

As client-side extensions it allows an application to place elements on a HTML form and respond to user events such as mouse clicks, form input, and page navigation. Server-side web-scripting languages provide developers with the capability to quickly and efficiently build Web applications with database communication. They became in the last decade a standard for dynamic pages. PHP being one of the most popular scripting languages was chosen for developing the system. Its main advantages are facilities for database communication. People are used to Web services which provide structured information, search engines, email application etc. The proposed system had to be similar to those services in its functionality and generality. Most timetable applications give a possibility to save schedules for particular classes, teachers and rooms as HTML code, but they do not allow interaction with its users. Creating a timetable is a process which involves often a lot of people working on it. There are not only timetable designers, but also teachers, who should be able to send their requirements and preferences to the database. This is to a high extent facilitated by a web application, which allows interaction between teachers, students and timetable designers. An example screen of the web timetabling application is presented in Figure 2.



APC - Automatyzacja Procesów Ciągłych
IP - Identyfikacja Procesów
MiS - Modelowanie i symulacja
MP - Miernictwo Przemysłowe
PO - Programowanie Obiektowe
RO - Rozpoznawanie Obrazów
TI - Technologie Internetowe
TSt - Teoria Sterowania
UA - Urządzenia Automatyki

Figure 2. The example screen of the Web application

Timetable Manager

It is hard to develop a fully functional user interface using only Internet technologies. Therefore VC++ was used to build the Timetable Manager, a program for timetable designers. The idea of the program was to simplify manual timetabling and to provide as much information as possible during this process. Operating on the data locally significantly increases performance during data manipulation, data manipulation is based on SQL queries on database. Well-known features – drag and drop can be implemented, layout is based on tree navigation. One of the most important feature of the timetamble manager is assistance during manual timetabling. Small timetables, which automatically show schedules for resources of a selected course can be freely placed by the user. During manual scheduling available timeslots are shown and constraint violations are explained by proper colors. Data can be saved in two ways:

- data is saved locally in dedicated file format,
- data is synchronized with remote database.

The system takes into account privileges of users and does not allow unauthorized change of data. An example screen of the Timetable Manager is presented in Figure 3.

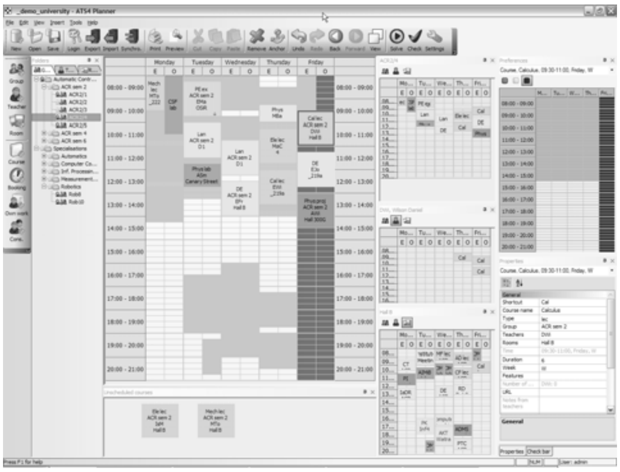


Figure 3. The example screen of manual assistance in Timetable Manager

Multi user support

Allowing user to work locally forces to develop of a data synchronization mechanism between locally changed data and remote database. The proposed mechanism is based on idea of versioning systems like CVS or SVN. But taking into account timetable data is much harder than text files, because of complicated relations between data.

The main advantages of this mechanism are following:

- simultaneously changes are allowed and in case of conflict possibilities discard changes or introduce them are given to the user,
- user have very quick access to all timetable without blocking them for other users,
- changes can be applied for a lot of data (e.g. through locally solver) ,
- if data are not changed by one user or user has no rights to change data, there updated without inform the user,
- default values for changes are chosen in such a way, that newest changes are taken or changes with higher level of rights.

Two actions are proposed to take care of integrity of the data:

Import/update (it is required if data are changed remotely and user want make export)

1. Assume unique index for each course and resource and date of the last change. Indexes of deleted resources are remembered in separate table. *maxIndex* – the greater value of all indexes.
2. Remember locally current state (*local_UT*) and a whole state of the last imported timetable (*last_ remote_UT).*
3. Select changes from database, which are newer than *last_ remote _UT.*
4. Introduce changes to the *last_remote_UT* and build *remote_UT.*
5. Indexes of local resource, which are greater than *last_remote_UT.maxIndex* are increased by *remote _UT.maxIndex - last_ remote _UT.maxIndex.*
6. Compare all data of the 2 timetables (*remote_UT* and *local_UT*), and check what kid of data was changed locally or remotely. Give user possibilities to accept or reject changes for data which change both locally and remote.
7. By default assume acceptance of the changes.
8. Replaced *last_remote_UT* with *remote_UT.*

Export/commit

1. Make Import to check changes and build *remote_UT.* Export is available if the *last_remote_UT* does not differ from *remote_UT.* Otherwise import is forced.
2. Compare *local_UT* with *remote_UT* based on the last change date to show user what changes will be exported
3. Assume default introduced changes to send them to database.
4. If some resources or courses are removed, store indexes with data in a special table.

Multi user support was the most desire feature of the whole timetable system. It can be solved by online working on database with multi-user access, transactions and locking tables. But this solution was rejected, because of low performance in case of simultaneous work of many users.

Automated timetabling based on Constraint Programming paradigm

The presented solver is written in ECLiPSe [ECL] using the Constraint Programming paradigm and replaced solver written in Mozart/Oz language. Main idea of the methodologies are similar to those widely presented in author's PhD thesis [Leg06], [Leg03]. The main idea of the solver were:

- effective search methods are customized for taking soft constraints into account during the search, based on the idea of value assessment [AM00]
- custom-tailored distribution strategies are developed, idea of constraining while distributing, which allowed to effectively handle constraints and search for 'good' timetables straight away,
- custom-tailored search methods were developed to enhance search effectiveness of timetabling solutions,
- integration of Local Search techniques into the Constraint Programming paradigm search enhanced optimization of timetabling solutions .

Additionally flexibility of the timetable definition was widened by the week definitions and dynamic resource assignment.

Week definitions

The idea of incorporating week definitions into the problem definition comes from the fact that scheduling an "odd" course will cause unused time in the "even" weeks and vice versa. This might cause long gaps between courses and could also render the problem unsolvable. To deal with this disadvantage we could prolong the scheduling period from one to two weeks to take account of courses with a longer cycle. This unfortunately has a drawback of doubling the domains of the courses' start variables and a necessity to add special constraints (to enforce the weekly scheduled courses happening in the same time during both weeks). The aforementioned solution would however not apply in some situations, for example in the case when some courses are required to happen only a few times in the semester or only in the second half of the semester. It would also increase the size of variable domains causing a great computational overhead. We can eliminate these drawbacks thanks to the introduction of week definitions. Week definitions are logical structures that group a certain number of time periods from the whole time-span. Referring to the previous examples a week definition of odd weeks would consist of weeks numbered <1 , 3 , 5 …15>, week definition of all weeks<1,2,3,…16> and so on , more examples below:

```
week_def{id:″A″,
  weeks:[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
}  ( all weeks )
week_def{id:″O″,   weeks:[1,3,5,7,9,11,13,15]}
  ( odd weeks )
week_def{id:″E″, weeks:[2,4,6,8,10,12,14,16]}   (
even weeks )
week_def{id:″SHO″, weeks:[9 ,11,13,15]}  (second
half of the semester odd weeks)
week_def{id:″F4W″, weeks:[1,2,3,4]}  (first four
weeks)
```

For certain pairs of week definitions we state whether they are in conflict, which corresponds to the fact that their sets of weeks have common elements. Basing on the example above we would say that conflicts are:

```
week_def_conflict{id1:”A”, id2:”O”}
week_def_conflict{id1:”A”, id2:”E”}
week_def_conflict{id1:”A”, id2:”SHO”}
week_def_conflict{id1:”A”, id2:”F4W”}
week_def_conflict{id1:”O”, id2:”SHO”}
week_def_conflict{id1:”O”, id2:”F4W”}
week_def_conflict{id1:”E”, id2:”F4W”}
```

Having defined week definitions we take from the input data assignment at least one of them to each course:

```
course {id:”act1”,start_time:SA1 duration:5, … ,
week_defs:[“O”] , … }
(“act1” taking place in odd weeks)
course   {id:”act2”,start_time:SA2   duration:7
week_defs:[“F4W”, “SHO”] , … },
(“act2” taking place in first four weeks of the
semester and in odd weeks of the second half of
the semester )
```

These information is used at the constraint setup phase. Pairs of courses which do not contain any conflicting definitions are excluded from the constraint setup, because they occur in different weeks and therefore there is no risk that they would require the same resources during the same time.

Pairs of courses that contain at least one pair of conflicting week definitions, are potentially competitors for the same resources during the same time and need to be taken under consideration during constraint setup.

The idea of week definitions is a universalized and convenient approach of handling courses which are exceptional and do not occur regularly within each time period.

Dynamic resource assignment

We have taken the approach that resources do not have to be instantiated at the phase of problem definition, which on one hand enforces a more complex programmatic approach but on the other better reflects the nature of real timetabling problems and also allows greater flexibility at search phase (possibility of balancing resource usage, moving courses between resources might lead to further optimization of the cost function).

Normally we would assume that a course requires a fixed set of resources to take place. That would be for example, a group of teachers, a group of classrooms and a group of students, all known and stated at the time of problem definition. We extend this model by enabling the user to state how many elements from a group of resources are required , without an explicit specification which ones should be used. This flexibility is achieved thanks to the definition and management of resource groups implemented in our XML interface and processed by the solver. The data structure is such that for every course we define resources. Resources are defined by a (theoretically unlimited) number of resource groups. Each group contains indexes, that correspond to certain resources and, as a property, a number of required resources.

The number of required resources can range from one to the cardinality of the group. When the number of required resources is maximal, all the resources within the group need to be used, but for any number below the maximum we are left with a choice of resources.

```
<Course>
…
<Resources>
     <Group required=2 >
       <Resource>teacher_32</Resource>
       <Resource>teacher_78</Resource>
       <Resource>teacher_93</Resource>
     </Group>
     <Group required=1 >
       <Resource>classroom_122</Resource>
       <Resource>classroom_123</Resource>
       <Resource>classroom_144</Resource>
       <Resource>classroom_215</Resource>
     </Group>
     <Group required=1 >
       <Resource>students_group_23</Resource>
     </Group>
     <… optionally more groups>
</Resources>
</ Course >
```

This structure is translated into resource variables list in each course.

```
Course   … , resource_variables_list:[Teacher1,
Teacher2, Classroom1, StudentGroup1] , …}
```

And domains of those variables present in the list

```
domain(Teacher1)=domain(Teacher2) = [teacher_32,
teacher_78, teacher_93 ]
domain(Classroom1)=[classroom_122              ,
classroom_123 , classroom_144 , classroom_215]
```

For every group of resources we create as many resource variables as number of required resources, and give each of them a domain of all resources in a group , then constrain them to be all-different (since we cannot use any resource twice in one course). For those groups where all resources are required, variables should get instantiated right away which corresponds to the model with fixed resources:

```
StudentGroup1 = students_group_23
```

What we need to ensure now is that any two courses do not use the same resource at the same time. This is achieved for instantiated resources by imposing a constraint that prevents courses from overlapping in time, for every pair of courses that use the same instantiated resource and are in conflict according to week definitions. It is sometimes possible to set up global constraints involving more than two courses that require the same resource but only if each pair in the group is in conflict according to their week definitions, which is not always the case.

What still needs to be handled are the uninstantiated resource variables with domains. To do this we impose a suspended test on every pair of courses that have at least one common resource in their resource variables domains and are in conflict according to their week definitions. The tests wait for instantiation of both resources that could potentially be the same, and checks if they are. If the test succeeds, the constraint that prevents the pair of courses from overlapping is imposed on the courses. The invocation of tests and consequently imposing of constraints happens at the search phase when resources get instantiated by the search algorithm.

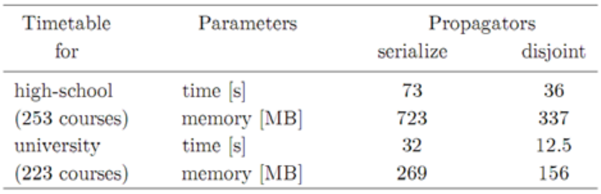
To enhance the constraint propagation it is useful to impose a second set of tests on the courses to ensure that the same resources are not chosen for courses that overlap in time. To achieve this , for each pair of courses that are in conflict according to their week definitions we impose a test checking whether the courses overlap (different conditions guarding for domain updates are acceptable here, domain bound changes as well as variable instantiation). If the test succeeds the all-different constraint is imposed on resource lists of the two courses stating that none of the variables in one list takes the same value as any variable in the other (since they can not use the same resources whilst overlapping in time and belonging to conflicting week definitions).

This second set of tests (considering courses' start times) is redundant. We notice that its declarative meaning is the same as for the first set of tests (considering resource variables) , but in the case when we proceed through the search tree both by instantiating start times for courses and resource variable, we get a better constraint propagation and avoid exploring some parts of search tree which do not contain a solution.

There is a need to use these suspended tests that set up constraints during search phase, because at the constraint setup phase we do not have the knowledge which activities will overlap in time or which will use the same resources therefore we need to wait for further instantiation of variables. This slight complication is the consequence of using dynamic resource assignment.

Results

The final results cannot be presented, because of the implementation stage of the whole system. Some results are taken from previous solver written in Mozart/Oz language for two small real problem – one from high-school and departure at the Silesian University of Technology. Results presented in Figure 4 shows that using a too complicated propagator can twice increase time and memory consumption.



Timetable for	Parameters	Propagators serialize	disjoint
high-school	time [s]	73	36
(253 courses)	memory [MB]	723	337
university	time [s]	32	12.5
(223 courses)	memory [MB]	269	156

Figure 4. Comparison of two types of no overlap constraints.

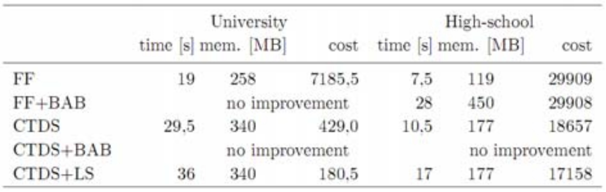
Schedule.serialize is a strong propagator to implement capacity constraints. It employs edge-finding, an algorithm which takes into consideration all tasks using any resource. This propagator is very effective for job-shop problems. However, for the analyzed cases this propagator is not suitable, because most tasks have frequently small durations and the computational effort is too heavy as compared with the rather disappointing results. FD.disjoint which although may cut holes into domains, must be applied for each two courses that cannot overlap. Those constraints enable also the handling of some special situations connecting with week definitions described in previous section.

Popular first-fail (FF) strategy was compared with custom-tailored distributed strategy (CTDS) based on constraining while distributing and choosing those values for variables, which have smallest assessment (assessment for value was increased when soft constraints were violated). Optimization was checked for popular branch-and-bound and idea of incorporation local search into constraint programming. This idea based on following steps after finding feasible solution:

1. Finds a course which introduced highest cost (e.g. makes gaps between courses)
2. Finds a second course to swap with the first one.
3. Creates a new search for the original problem from memorized initial space.
4. Instantiates all courses (besides these two previously chosen) to the values from solution. It can be made in one step because they surely do not violate constraints.
5. Schedules first course in the place of the second one.
6. Finds the best start time for the second course.

7. Computes the cost function. If it has improved, the solution is memorized, else another swap is performed.

Results of comparisons are presented in Figure 5.



	University			High-school		
	time [s]	mem. [MB]	cost	time [s]	mem. [MB]	cost
FF	19	258	7185,5	7,5	119	29909
FF+BAB		no improvement		28	450	29908
CTDS	29,5	340	429,0	10,5	177	18657
CTDS+BAB		no improvement			no improvement	
CTDS+LS	36	340	180,5	17	177	17158

Figure 5. Comparison of two types of distribution strategy and optimisation methods.

Conclusion and future work

Presented system describing comprehensive approach to real-world University Timetabling problem is still during implementation at the Silesian University of Technology. Most of the parts system has been already implemented, but it is still not used in full range. Multi-user paradigm has been already implemented and tested. It is one of the most important feature appreciated by the user, which use nowadays only manual assistance of the presented system. Authors plan test different methodologies based on Constraint Programming and Local Search after gathering data from whole university. The different search methods will be tested similar to Iterative Forward Search presented in [M05].

References

[AM00] S. Abdennadher and M. Marte. *University course timetablingusing constraint handling rules*. Journal of Applied Artificial Intelligence, 14(4):311–326, 2000.

[ECL] The ECLiPSe Constraint Programming System, http://eclipse.crosscoreop.com/

[Leg03] W. Legierski. *Search strategy for constraint-based class-teacher timetabling*. In Practice and Theory of Automated Timetabling IV, volume 2740 of Lecture Notes in Computer Science, pages 247–261. Springer-Verlag, 2003.

[LW03] W. Legierski and R. Widawski. *System of automated timetabling*. In Proceedings of the 25th International Conference Information Technology Interfaces ITI 2003, Lecture Notes in Computer Science, pages 495–500, 2003.

[Leg06] W. Legierski. Automated timetabling via Constraint Programming, PhD Thesis, Silesian University of Technology, Gliwice, 2006.

[Mar02] M. Marte. *Models and Algorithms for School Timetabling A Constraint-Programming Approach*. PhD thesis, Ludwig-Maximilians-Universitat Munchen, 2002.

[M05] T. Muller. *Constraint-based Timetabling*. PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, 2005.

[PB04] S. Petrovic and E.K. Burke, Edmund K, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Chapter 45: University Timetabling*,CRC Press,Edt: J. Leung, 2004

[RL01] Oliveira E. Reise L.P. *A language for specifying complete timetabling problem*. In Practice and Theory of Automated Timetabling III, volume 2079 of Lecture Notes in Computer Science, pages 322–341. Springer-Verlag, 2001.

[Rud01] H. Rudova. *Constraint Satisfaction with Preferences*. PhD thesis, Masaryk University Brno, 2001.

[Sch95] A. Schaerf. A survey of automated timetabling. In Wiskunde en Informatica, TR CS-R9567. CWICent, 1995.

[Wer86] J. Werner. *Timetabling in Germany: A survey*. Interfaces, 16(4):66 74, 1986.

Opmaker2: Efficient Action Schema Acquisition

T.L.McCluskey, S.N.Cresswell, N. E. Richardson and M.M.West
School of Computing and Engineering
The University of Huddersfield, Huddersfield HD1 3DH, UK

Abstract

The problem of formulating knowledge bases containing specifications of dynamic knowledge is a barrier to the widespread uptake of AI planning. Machine learning has been used with some success in the past, but the inputs required are either too detailed, or the learning process has required many examples. Further, learning has been confined to propositional actions or parts of actions such as preconditions. The field of ontological engineering has had an impact on the wider community in that application ontologies (which contain "static" structural knowledge of applications) are becoming widespread. Here we introduce a methodology that is based on the existence of a strong structural model of an application. Using a small number of user training sequences, we illustrate how the method can induce action schema and compound methods. To do this we extend GIPO's Opmaker system so that it can induce actions from training sequences without intermediate state information and without requiring large numbers of examples. This method shows the potential for considerably reducing the burden of knowledge engineering, in that it would be possible to embed the method into an autonomous program (agent) which required to do planning. We illustrate the algorithm as part of an overall method to induce structured domain model, and comment on initial results that show the efficacy of the induced model empirically.

Introduction

The problem of formulating knowledge bases containing specifications of dynamic knowledge is a barrier to the widespread uptake of AI planning. Current high profile applications such as the use of planning technology within NASA's Mars Rover require persistent resources comprising of teams of highly skilled knowledge engineers, In particular, a problem facing AI is to overcome the need to hard code and manually maintain action schema within agents (a problem which limits their autonomy). It is possible to use learning techniques to help overcome the problem, eg using tools which induce actions or methods from examples. One method is to embed agents with the ability to induce the detailed specification of action schema from example planning traces, possibly supplied by a trainer. Planning traces are an ordered set of action instances, where each action instance is identified by name plus the object instances that are affected or are necessarily present but not affected, by action execution. This is the kind of information normally expected as a solution to planning problems.

In this paper we describe the results of an investigation into (re)constructing action schema and planning heuristics from training sessions which compose of a handful of action traces. The main result is that it is possible for an agent to induce detailed specifications of action schema from single action traces automatically, without requiring intermediate state information for each training example. The trade-off is that the agent's domain description should contains invariants describing object relations and object states. The induced actions are detailed enough for use in planning engines. We present an algorithm for generating such domain models, and show how the primitive action schema can be built up into domain models.

In our previous work we have shown how 'flat' domain actions can be induced from examples. Actions can be induced using *Opmaker* (McCluskey, Richardson, & Simpson 2002) which has been embedded interactively in *GIPO* (Simpson *et al.* 2001), (Simpson 2005). *GIPO* aids domain construction, offering editors, validation tools, a graphical life-history editor and planning tools. Output from *GIPO* is the completed and validated domain being modelled in a variant of GIPO's internal language *OCL* (Liu & McCluskey 2000) or PDDL. Here we extend GIPO's Opmaker system so that it can induce actions from training sequences and its static object model alone, without intermediate state information and without requiring large numbers of examples. This considerably reduces the burden of knowledge engineering, so that a program (agent) can perform knowledge acquisition rather than it occurring through a human-driven process supported by a tool such as GIPO.

The rationale for setting up this problem is as follows. The acquisition / refinement of factual or static knowledge by agents is relatively straightforward. In the context of the internet and open systems, it is not unreasonable that an agent can acquire and refine such knowledge with some degree of autonomy. The rapid expansion of globally accessible ontologies within standard formats such as OWL, support the notion that intelligent agents will have access to factual knowledge. In contrast, the amount of effort needed to encode bug free, accurate action specifications and planning heuristics, and to maintain them, is significant. A necessary precondition of the use of current automated planning technology is that there exists a detailed action specification, and in many cases, heuristic knowledge. Hence we can ask the question: for every agent that can perform planning, must we hand code and hand maintain its action descriptions? No, if agents are to achieve this kind of autonomy, then they should be capable of learning and refining action knowledge and heuristics.

The Learning Problem

The general situation is one where an agent needs to perform reasoning about actions to achieve a desired goal, and in particular perform plan generation within an environment that it has knowledge of. Actions are real world operations that change the state of object(s) in the world in some way. The agent has knowledge of objects, and collections of similar objects making up distinct classes. It knows the possible states of a typical object of each class. It has knowledge of existing plans that other agents, or a trainer, has used. These plans are written in terms of verbs and affected objects (pick up block A with gripper B, lift up wheel A with jack B). Additionally, the agent is assumed to have axioms describing a naive physics of the world. However, the agent has not an explicit specification of actions in such a way that it can reason about their synthesis (or the agent does have such a specification but needs to refine, maintain or evolve it).

Given this situation, the learning problem is to induce a full parameterised specification of actions which can be used to do planning; and to induce heuristics which can be used to make the reasoning involved in the planning computationally tractable. Further, the agent should be able to *refine* any existing parameterised specification of actions, and heuristics, that it currently holds. The action specifications should be detailed enough so that they can be input to mainstream planning technology as epitomised by competitors in the IPC (the bi-annual international planning competition).

A Formulation of the Problem

We formulate the learning problem as follows:

INPUT: Assume the input to the learning problem is a 'model' of the world, and a set of training sequences, given as follows:
1.1 - there are a number of classes each containing a set of objects, each object belongs to one set (called a sort)
1.2 - each object of each class may be related to objects of other classes, and have property - value relationships with set of basic values (boolean or scalar). The relations and properties are defined in the usual way using predicates.
1.3 - each object of each class at a moment in time has a fixed 'state'. This state is defined by its relationship with other objects and/or the value of properties. There are a small, finite number of states for each object class.
1.4 - there is a set I of invariants relating the predicates given above. Informally, a set is adequate if any 'common sense' inference can be made from them, such as normal inferences about spatial relations.
1.5 a set of training plans of the form

(initial state, final state)
$name_1\ p_1, o_1$
$name_2\ p_2, o_2$
..
$name_q\ p_q, o_q$

$name_1..name_q$ are the names of the q actions in the training plan, and they are assumed to transform the initial state into the final state. Here $p_1, p_2, ..p_q$ are each lists of object names (they could be null) of unchanging or 'prevail' objects required by an action, and $o_1, o_2, ..o_q$ are each lists of object names affected by the execution of the action. Each of the list of prevail objects must be present in some state, but that state does not change during action execution.

- a (possibly empty) set of existing action schema. Within this formulation, action schema are parameterised object transformations.

OUTPUT a set of action schema that - is consistent with the static domain model components (1.1 -1.4); - can be instantiated into the training plans (1.5) supplied, and will transform the initial state into the final state heuristics derived from the training plans that can be used to guide a planner

Method

The learning method is specified by the algorithm description in Figure 1. In outline, the method is:

(i) use a set of heuristics and inferences to track the changing states of each object referred to within a training example, taking advantage of the static, object-state information and invariants within the domain model. Infer full details of object transitions for each

```
program $Opmaker2$
In partial domain model
In training sequence $SEQ$ with N actions, and each $e \in SEQ$ has components:
$e.Name$, $e.prevail$, $e.changing$ = name, unchanging objects, changing objects,
Out parameterised action descriptions and HTN methods
Definitions:
$O.c$ = current state of an object $O$
$O.s$ = sort of object $O$
$O.f$ = final state of an object $O$
$S^g$ = state class of any $ground\ state\ S$
$O^g$ = a distinct parameter which ranges through the sort of $object\ O$
$X_s$ = set of all sorts of parameters and objects in expression $X$
for each $e$ in $SEQ$ do
    Form $P$ = list of $O^g$ for all $O$ in $e.prevail \cup e.changing$;
    for each $O$ in list $e.preval$ do
        store component of the prevail $(O.s, O^g, O.c^g)$
    end for
    for each $O$ in list $e.changing$ do
        if $O$ is not affected by actions in the rest of $SEQ$
        then let $X = O.f^g$
         else choose $X$ from the state classes of $O.c$ such that
             $X \neq O.c^g$ and $P_s$ contains $X_s$
         store transition $T = (O.s, O^g, O.c^g \Rightarrow X)$
         match free vars in $T$ with those in $P$
     end for
     form actions from cross-product of all stored transitions
     such that the actions are consistent with invariants
end for
produce a method from the sequences of actions as in Opmaker.
procedure match free vars in $T$ with those in $P$
repeat
    for each parameter $x$ in transition $T$, $x \neq O$,
        choose a parameter $y$ in $P$ to match with
        $x$ such that $y \neq O$, $sort(x) = sort(y)$,
    end for
until parameter match set is consistent
end
```

Figure 1: Outline Design of the *Opmaker2* Algorithm

dynamic object.

(ii) use the techniques of the original Opmaker algorithm (McCluskey, Richardson, & Simpson 2002) to generalise object references and create parameterised operator schema from the specific object transitions extracted in (i) from the training examples.

To illustrate the main innovations of the method, we will use an example walk-though taken from our empirical evaluation involving an extended tyre-change domain. Assume a training sequence *SEQ* is input into *Opmaker*2 and this has components as follows:

name: do_up; prevail: wrench0,jack0, trim1; changing: hub1,nuts1
name: jack_down; changing: hub1,jack0
name: tighten; prevail: wrench0,hub1,trim1; changing: nuts1
name: apply_trim; prevail: hub1; changing: trim1,wheel5

This illustrates a short procedure for making a car wheel ready for operation once it has been hung on to an appropriate wheel hub. Informally, do_up is the operation of putting the nuts on the hub of a wheel when it is jacked up. The names such as wrench0, hub1 are references to actual objects. The prevail objects have to be necessarily present in a particular state but remain unaffected ('wrench0' is available, 'jack0' is jacking up the wheel, 'trim1' is hub1's wheel trim and has to have been removed). These objects need to be in particular states for the action to execute, and those states 'prevail' or stay the same during execution of the action. The 'changing' objects change state (hub1 becomes fastened up, the nuts1 are fastened up).

To illustrate some of the definitions in Line 1 of the algorithm in Figure 1, we have components of an object as follows:

$hub1.c =$ [unfastened(hub1), jacked_up(hub1,jack0)]
$hub1.f =$ [on_ground(hub1), fastened(hub1)]
$hub1.s =$ hub

Examples of other operations are (h and j are parameters):

$hub1.c^g =$ [unfastened(h), jacked_up(h,j)]
$hub1.c_s =$ [hub,jack]

Line 2 iterates through all the training examples. For the first training example, the problem is to determine what the new states are of hub1 and nuts1.

In Line 3, let $P =$ [w, j, t, n, h]. In Lines 4-6, the prevail components are got from the current state classes of wrench0, jack0 and trim1, as in the original Opmaker algorithm. The loop starting on line 7 is intended to determine the destination of each object that is changed by the action being learned. hub1 is the first changing object. From the given partial definition of the domain, it has four state classes which we name S1-4:

S1 = [on_ground(h),fastened(h)],
S2 = [jacked_up(h,j),fastened(h)],
S3 = [free(h),jacked_up(h,j),unfastened(h)],
S4 = [unfastened(h),jacked_up(h,j)]

hub1's current state is not necessarily its final one, as in the training sequence it is referred to again (in the second of the sequence, *jack_down*) as a changing object. Hence line 10 is executed. X cannot be S4 (since this is currently the generalisation of the object's current state, and the object has to change state class). In Line 11 P_s (= [wrench, jack, trim, nuts, hub]) contains all the sorts in each of state classes S1,S2 and S3, and so this does not narrow down the choices. Hence 3 transitions are stored:

(hub, h, [unfastened(h),jacked_up(h,j)] → [on_ground(h),fastened(h)])
(hub, h, [unfastened(h),jacked_up(h,j)] → [free(h),jacked_up(h,j),unfastened(h)])
(hub, h, [unfastened(h),jacked_up(h,j)] → [jacked_up(h,j),fastened(h)])

Iteration of line 7 with object *nuts*1 occurs next. It has three states:

T1 = [tight(N,h)]
T2 = [loose(N,h)]
T3 = [have_nuts(N)]

This leads to 2 possible transitions:

(nuts, N, [have_nuts(N)] → [tight(N,h)])
(nuts, N, [have_nuts(N)] → [loose(N,h)])

and hence 6 possible induced action schema (line 15). These six options are then checked for consistency with the domain invariants which are shown in Figure 2. The conjunction of state constraints in both the LHS and RHS of transitions of the newly formed action schema must be consistent with these invariants. In cases where they are not, the action schema is discarded.

This reduces the number of options to a single action schema. Processing of the other 3 actions in the training sequence leads to a single interpretation of state changes, as the changing objects involved are all in their final states, and hence 3 more generalised action schemas are generated. Finally, a hierarchical method is generated (line 18) by combining the 4 action schema in a similar fashion to the original Opmaker system (McCluskey, Richardson, & Simpson 2002).

Experiments and Results

The method has been implemented and merged with the original Opmaker system. We are using the same experimental approach as we used to test the original system:

- We hand-craft training sequences from a range of domains selecting actions that will build sensible methods for that domain.
- We use Opmaker2 to induce actions and hierarchical (HTN-type) methods from the training sequences.
- Using standard planners, we compare performance using old hand-crafted action schema to the use of induced schema.

Success will be judged using the following criteria:

- If a valid set of unique new actions is defined as actions that can solve the same problems the original training sequences were aimed at, can *Opmaker*2 induce these without having to encode a great deal of invariants into the domain models?
- Is it more efficient in terms of effort time to construct a domain using *Opmaker*2?
- Is it at least as efficient, in terms of planning time, to reach goals using *Opmaker*2 defined actions and methods?

Up to now we have experimented with 2 domain models: the extended tyre world, and the hiking domain (see http://planform.hud.ac.uk/gipo/ for details of these).

Since induction sequences deliver several actions and a single method, initial sequences were tailored to pro-

1. Equivalence between hub *fastened* and nuts *tight/loose* on hub.

   $$\forall\, H{:}hub\ .\ [fastened(H) \iff \exists\, N{:}nuts\ .\ (tight(N,H) \vee loose(N,H))]$$

2. Equivalence between *jack_in_use* and *jacked_up.*

   $$\forall\, H{:}hub\ .\ \forall\, J{:}jack\ .\ [jack\_in\_use(J,H) \iff jacked\_up(H,J)]$$

3. Equivalence between hub not *free* and *wheel_on* hub.

   $$\forall\, H{:}hub\ .\ [\neg free(H) \iff \exists\, W{:}wheel\ .\ wheel\_on(W,H)]$$

4. Equivalence between *trim_on_wheel* and *trim_on.*

   $$\forall\, T{:}wheel\_trim\ .\ \forall\, W{:}wheel\ .\ [trim\_on\_wheel(T,\, W) \iff trim\_on(W,\, T)]$$

5. Only a single set of nuts can be on a hub.

   $$\forall\, H{:}hub\ .\ \forall\, N_1{:}nuts\ .\ \forall\, N_2{:}nuts\ .\ \left[\begin{pmatrix} (tight(N_1,H) \vee loose(N_1,H)) \\ \wedge \\ (tight(N_2,H) \vee loose(N_2,H)) \end{pmatrix} \Rightarrow (N_1 = N_2) \right]$$

6. Only a single wheel can be on a hub.

   $$\forall\, H{:}hub\ .\ \forall\, W_1{:}wheel\ .\ \forall\, W_2{:}wheel\ .\ \left[\begin{pmatrix} wheel\_on(W_1,H) \\ \wedge \\ wheel\_on(W_2,H) \end{pmatrix} \Rightarrow (W_1 = W_2) \right]$$

7. Domain constraint: If nuts are tight on a hub then the hub must be on the ground.

   $$\forall\, H{:}hub\ .\ [(\exists\, N{:}nuts\ .\ tight(N,H)) \Rightarrow on\_ground(H)]$$

8. Domain constraint: if a trim is on a wheel, then the wheel is on a hub and the nuts are tight.

   $$\forall\, W{:}wheel\ .\ \exists\, T{:}wheel\_trim\ .\ \left[\begin{array}{c} trim\_on\_wheel(T,\, W) \Rightarrow \\ (\exists\, H{:}hub\ .\ wheel\_on(W,H)) \wedge (\exists\, N{:}nuts\ .\ tight(N,H)) \end{array} \right]$$

Figure 2: Invariants encoded in the Extended Tyre World

duce a meaningful method, and sufficient initial sequences were composed to cover all the major sub-tasks that could be required by the domain. In each case the agent began by knowing domain knowledge but had sketchy or non-existent facts about its potential actions. For the Extended Tyre World we devised 7 sequences of between 2 and 5 actions in length. After adding 8 invariants to the domain we induced a set of actions and methods and using these we produced a domain with 22 actions and 7 methods. The new version was tested over 8 tasks in two ways - firstly using just actions in the planning and secondly using either just methods, or a combination of methods and actions. To illustrate the results, two of the actions that were induced from the running example were as follows:

```
operator(jack_down(Hub1,Jack0),
[],
[sc(hub,Hub1,[jacked_up(Hub1,Jack0),
              fastened(Hub1)] =>
  [on_ground(Hub1),fastened(Hub1)]),
sc(jack,Jack0,[jack_in_use(Jack0,Hub1)] =>
  [have_jack(Jack0)])], []).
```

```
operator(tighten(Wrench0,Hub1,Nuts1,Trim1),
[se(wrench,Wrench0,[have_wrench(Wrench0)]),
se(hub,Hub1,[on_ground(Hub1),fastened(Hub1)]),
se(wheel_trim,Trim1,[trim_off(Trim1)])],
[sc(nuts,Nuts1,[loose(Nuts1,Hub1)] =>
  [tight(Nuts1,Hub1)])], []).
```

Where just actions were used in planning, plan times for short plans of up to 10 to 12 actions were about the same as for the hand-crafted version of the domain. For plans longer than 12 actions both versions took increasingly long times to solve. However where methods or combinations of actions and methods were used plan times were significantly shorter. The full planning problem for this extended domain is defined to be: "A car is found to have two flat tyres, one is found to be flat and can be fixed by use of the pump, whilst the other is punctured and requires the full tyre change described in the previous version of the domain". Using just actions no solution was found to this problem after 36 hours but using methods and just a few actions a correct solution was found after 11 seconds.

Experimentation with the hiking domain is at an earlier stage. As yet no invariants have been added to the domain. Without these we do not get unique sets of example material for induction but already we have seen actions generated. We identified 5 potential methods for this domain and for four of these we obtained example sets of no larger than 6. However the fifth generated 28 example sets so either a set of invariants will be added to the agent's knowledge, or we will use theory refinement to reduce the example sets further.

From the results obtained so far we can conclude that an agent, given a 'working stock' of potential action sequences, and having domain knowledge and a 'belief' about the states of objects it 'knows' about will be able to generate its own examples and use them to supply itself with parameterised actions to suit every possible object combination. Since methods can be formed from the action sequences the agent should be able to plan efficiently and autonomously.

Related Work

The authors of (Garland, Ryall, & Rich 2001) have developed a system (Collagen) which learns task models from examples. Their work is similar to ours in that they show orderings of the task to achieve the task and these contain both primitives and non-primitives. In (Wu, Yang, & Jiang 2005) the authors describe ARMS, a system in which operators are learned without the need for user intervention. However ARMS requires many training examples containing valid solution sequences, and presently is capable of inducing only 'flat' domains.

Our work is also aimed at learning domains containing both action schema and hierarchical schema (methods) encapsulating several schema. Practical planning domains are based on 'hierarchical task network' (HTN) decomposition. The chief difference between the HTN paradigm and classical domains is that in the former 'compound' tasks can be decomposed into the simpler 'tasks' particular to classical domains. However HTNs can be difficult to construct manually and authors have worked in producing these using methods from machine learning. In (Erol, Hendler, & Nau 1996) the authors argue that HTN operators are more expressive than those of classical domains as well as being more efficient. Theoretical underpinning for 'High Level Actions' (HLAs) is presented in (Marthi, Wolfe, & Russell 2007). Each HLA admits one or more *refinements* into sequences of actions, where an action might be high level or primitive. The paper introduces a provably sound and complete algorithm which is implemented using a STRIPS-like language. The algorithm takes advantage of 'sound and complete' descriptions and, if successful, returns a primitive refinement of some high-level plans that achieves the goal set from the initial state.

In (Nejati, Langley, & Konik 2006) the authors describe how they induce *teleoreactive logic programs* from expert traces. The teleoreactive programs index methods by the goals they achieve. They use methods derived from explanation based learning to chain backwards from the end result of the sample trace. The explanation structure thus obtained is retained to produce new hierarchical structures. The method is applied to 'Depots' which involves crates that can be loaded into trucks and stacked. However the domain so constructed resulted in the successful solution of very few problems.

Further theoretical work on HTN planning is presented in (Ilghami *et al.* 2005). This paper introduces a formalism whereby situations are modelled where general information is available of tasks and sub-tasks, together with some plan traces but there are no details. In the early work all information about methods was required except for the preconditions. This limitation is overcome in later work by the same group (Ilghami, Nau, & Munoz-Avila 2006) a new algorithm 'HDL' (HTN Domain Learner) is presented which learns HTN domain descriptions from plan traces. Between 70 and 200 plan traces are required to induce the descriptions.

HTN-MAKER is presented in (Hogg & Munoz-Avila 2007). This receives as input a STRIPS domain model, a collection of STRIPS plans and task definitions and produces an HTN domain model. The experimental hypothesis is that after a few problems have been analysed an HTN domain model will be ultimately obtained able to solve most solvable problems. A version of the logistics-transportation domain is chosen for the experiment and good results are obtained. However these good results are not replicated for the blocks-world domain. One problem is the large number of methods which have to be learned, where one method might subsume another. They suggest choosing the most general method where this is the case. Another problem is for the planner to use methods in an infinitely recursive manner.

Conclusions

Our work and the results reported here depend on a structured view of domain knowledge about objects being available. Whereas in propositional, classical planning states are fairly arbitrary sets of propositions, we assume that the space of states is restricted in that objects are pre-conceived to have a fixed set of plausible states. Within this framework, we have described a method for inducing action schema that advances the state of the art in that it requires no intermediate state information, or large numbers of training examples, to induce a valid action schema set. Further, our preliminary results show that the hierarchical methods induced with the action schema can lead to more efficient domain models.

Opmaker2 is an improvement on Opmaker in that the latter requires intermediate state information during learning. Opmaker2 automatically infers this intermediate state information and then proceeds in the same fashion as Opmaker and induces the same operator schema. Opmaker2 can logically be seen as a superset of Opmaker, where the extra functionality in Opmaker2 removes the need to ask the trainer for more information.

Our experiments with the "Hiking Domain" show that further development needs to be made to the Opmaker2 algorithm so that it can cope with domains with "static" knowledge.

References

Erol, K.; Hendler, J.; and Nau, D. S. 1996. Complexity Results for HTN Planning. *Annals of Mathematics and Artificial Intelligence* 69–83.

Garland; Ryall; and Rich. 2001. Learning hierarchical task models by defining and refining examples. In *Proceedings of the First International Conference on Knowledge Capture.*

Hogg, C., and Munoz-Avila, H. 2007. Learning Hierarchical Task Networks from Plan Traces. In *Proceedings of the ICAPS'07 Workshop on Artificial Intelligence Planning and Learning.*

Ilghami, O.; Nau, D. S.; Muoz-Avila, H.; and Aha, D. W. 2005. Learning preconditions for planning from plan traces and HTN structure. *Computational Intelligence* 21(4):388–143.

Ilghami, O.; Nau, D. S.; and Munoz-Avila, H. 2006. Learning to do htn planning. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling*, 390 – 393.

Liu, D., and McCluskey, T. L. 2000. The OCL Language Manual, Version 1.2. Technical report, Department of Computing and Mathematical Sciences, University of Huddersfield .

Marthi, B.; Wolfe, J.; and Russell, S. 2007. Semantics for High-level Actions. In *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS 2007.*

McCluskey, T. L.; Richardson, N. E.; and Simpson, R. M. 2002. An Interactive Method for Inducing Operator Descriptions. In *The Sixth International Conference on Artificial Intelligence Planning Systems.*

Nejati, N.; Langley, P.; and Konik, T. 2006. Learning hierarchical task networks by observation. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, 665–672. New York, NY, USA: ACM Press.

Simpson, R. M.; McCluskey, T. L.; Zhao, W.; Aylett, R. S.; and Doniat, C. 2001. GIPO: An Integrated Graphical Tool to support Knowledge Engineering in AI Planning. In *Proceedings of the 6th European Conference on Planning.*

Simpson, R. M. 2005. Gipo graphical interface for planning with objects. In *Proceedings of the International Conference for Knowledge Engineering in Planning and Scheduling.*

Wu, K.; Yang, Q.; and Jiang, Y. 2005. Arms: Action-relation modelling system for learning acquisition models. In *Proceedings of the First International Competition on Knowledge Engineering for AI Planning.*

Feasibility Criteria for Investigating Potential Application Areas of AI Planning

T. L. McCluskey
School of Computing and Engineering
The University of Huddersfield, Huddersfield HD1 3DH, UK
lee@hud.ac.uk

Abstract

In this paper we address the problem of deciding whether it is feasible to apply AI planning technology (involving currently available planning engines) to an application area. We develop some criteria based on motivation, technological infrastructure and knowledge engineering aspects of an application, and we go on to apply these criteria to two application areas. The criteria both help to evaluate the overall feasibility, and in cases where development continues, help us to focus on the parts of the application which are likely to be most troublesome.

Introduction

In recent AI conferences (ICAPS, ECAI) there have been a number of workshops devoted to AI planning applications, and ICAPS itself gives an award to 'best application' paper. While many applications tend to be in AI-rich environments such as Space Technology, there is a growing body of applications from a wider range of areas. A notable example is the SIADEX (Fdez-Olivares *et al.* 2006) project, developing tools for helping people to manage forest fire fighting resources. Several other notable applications were described in the recent ICAPS 'Moving Planning and Scheduling to the Real World' workshop (Myers *et al.* 2007). However, we still appear to be very far away from the point where automated planning technology can be franchised to the software engineering community.

Our work is motivated by investigations into the use of AI planning in large-scale *control* applications. Automated assessment and prediction via monitoring and modelling is quite well developed in these kinds of applications, but there is a need to develop software support that enables active decision support or even autonomous control eg in water/flood control (Rob 2007), or road transport network control (Various 1999). However, how feasible is the use of AI planning tools within such an application area? How could we evaluate an application in terms of whether it can benefit from AI planning technology, and how can we determine what areas of the application would cause the most problems? In this paper we explore the characteristics of an application area that make the application of AI planning feasible. To motivate the discussion, we use two particular applications from the Transport and Water Management service industries respectively. These are wide ranging, complex, involve many stake holders and organisations, and have allied research and development areas.

This endeavour has much in common with the general area of business process change through the introduction of new technology, and in particular the introduction of knowledged-based AI technology. The potential problem areas in the application of automated planning are in some cases similar to challenges already well known when implementing KBS systems. These include the 'knowledge bottleneck' - the difficulty of knowledge elicitation and formulation, the availability of experts and expertise, and the verification, validation, and maintenance of knowledge bases. The subject of this paper can be taken in the context of the well known reasons for failure of early KBS, to do with their brittleness and stand-alone nature. However, applications of automated planning can also take advantage of the more recent developments that alleviate the 'knowledge bottleneck': the development of shared ontologies and globally accessible knowledge, and the development of standard, tool support environments for the engineering of knowledge. For a discussion of the similarities and distinguishing features between knowledge engineering for AI planning and KBS, the reader is referred to section 7 of PLANET's Roadmap (Biundo *et al.* 2003).

In this paper we address the problem of evaluating the feasibility of applying AI planning technology, by devising a set of evaluation criteria based on motivation, technological infrastructure and knowledge engineering aspects of an application. To both illustrate and evaluate the usefulness of these criteria we use them to investigate the feasibility of applying automated planning technology to the applications. For each feature we rank it as low, medium or high, indicating its contribution towards an overall feasibility factor. We conclude with a short discussion of the use of the criteria.

Feasibility Evaluation Criteria

We assume that an 'application area' has been identified, and there is a prima facia case for the use of automated planning within it. In the case of the two applications considered below: (i) road network management: planning can be used for drawing up plans to ease congestion or alleviate the effects of incidents (ii) flood prevention and management: planning may be applied to form plans for evacuations. Given this context, we postulate a number of key questions that need to be considered in evaluating the feasibility and effectiveness of utilising automated planning. We group them into 3 areas:

1.Motivation Factors

Motivation factors include the fundermental and underlying reasons for the introduction of planning technology. If a current system delivers an optimum solution, or a subset of stakeholders are satisfied with the operation of the system using current technology, then the motivation may be too weak. An example of low motivation is where there may be pressure to introduce advanced technology for its own sake, rather to satisfy a perceived need.

Overall the questions that should be asked include: are there compelling reasons for the introduction of technology: is it likely to deliver a step change is quality of service being offered eg increased reliability of plans, correctness of plans, real-time speed-up in the generation, and execution or distribution of plans? Will AI planning enable a significantly more cost-effective solution to some perceived problem?

2.Technological Context and Human Factors

Feasibility with respect to the application's context increases if there is already a high level of technological development within the service or industry. In human controlled systems there are several well defined phases in control: understanding what is happening in the system, evaluating that understanding (is there a problem?) and generating an effective plan to help alleviate the problem. The introduction of planning technology is more likely to be feasibility if IT is already heavily used in the collecting, processing and interpretation of data, and in providing support for the current decision processes. If the data collected has an uncertain interpretation, or is incomplete, then the feasibility factor is lessened.

Given the nature of the technological change, it is helpful if there already exists experimental platforms to support the introduction of new technology. Typically this would comprise of historical data and a simulation system which can be used to investigate the effectiveness of techniques off-line.

Feasibility also depends on human factors: if key stakeholders are unwilling to accept the kind of autonomy delivered by automated planning then it will not be feasible. An example is where the current problem owners contract out the planning task to a third party. The third party is not necessarily going to he a willing partner in the venture if the new technology threatens that contract; the third party, however, may hold knowledge that is necessary for success. In summary, the key questions are:

Existing technological infrastructure:

Within the computer systems that are currently being used in the potential application area, are sophisticated systems used extensively for management information, and/or for decision support? What is the level of technological take-up in the area? Are the current systems stand-alone or fully interoperable?

Data availability and quality:

Is there a ready supply of data to supply state information on the observed system? Is the data in high level (information extracted) form, or is it in a very low level (eg numerical) form? Is the data trustworthy or does it contain a significant amount of uncertainty? Can data be extracted from a standard data interface? Is there historical data and/or a simulation environment that can be used to test new technology off line?

Human Factors:

Are the problem owners (the current providers of solutions) and other stakeholders open and supportive of innovation to help improve their methods and systems?

3.Knowledge Engineering Factors

There is a well known characterisation of AI planning technology that it requires the pre-engineering of a specific database of actions, heuristics etc. The task of engineering knowledge into such a particular form is itself made feasible by the presence of a number of factors, such as: existing high level formalisations of the domain, existing high level formalisations of plans, or the existence of similar planning domain models. These factors are very relevant in knowledge engineering for KBS in general, as it is well known that if all the expertise lies solely in the brains of experts, then the amount of effort involved in knowledge elicitation and knowledge formulation can be very high indeed. Applications where there are existing encodings of actions and plans are thus very attractive. Hence, if knowledge of the current planning process in the application area is not in written form, or there are no examples of precisely encoded plans, then the feasibility is low, or at least the amount of resource needed to create a domain model and planning heuristics may be prohibitive. The key questions to consider are:

Closeness to previous applications:

Is the application area close or analogous to a previous defined planning benchmark, or a current fielded system? Can parts of domain models or previously engineered constraints be re-used?

Procedure formalisation in the problem area:

Are there existing encoded, formalised or written-down procedures or plans? Is the current system managed by experts using their own experience, or do they have recourse to manuals and training aids? Are there readily available examples of the kinds of plans that are needed to be generatored?

Appropriateness for AI planning solution:

How far does the construction of a plan fall into the classic definition of generating orderings of instantiated action schema to achieve goals or decompose tasks? Does plan generation involve a great deal of uncertainty, mixed discrete and continuous variables, or large amounts of human skill?

Application Area: Road Network Management

Description

Road network management (RNM) relies on complex, integrated systems to meet increasing requirements upon the road network specified within policy documents from central and local government. The responsibility for managing the road infrastructure in the UK rests with the Highways Agency (HA) for the motorway and trunk road network and the Local Authority (LA) for the urban network. Short term traffic events, such as road works, accident, adverse weather conditions, occurring on either the motorway or urban network can have devastating affects on one another. Currently human operators respond to this kind of problem using their expert knowledge, but their effectiveness is limited as they have to interpret complex information fed to them, decide on which of an array of actions to take, and deal with the interface between urban and motorway traffic control. Within the UK there is a duty on LAs to manage their traffic networks efficiently and reduce traffic pollution. Clearly there is a need to develop systems that will support the road network operators objectives when they try to tackle congestion or other problems, such as excessive fuel emissions, in an increasingly complex environment.

Evaluation using Criteria

Motivation Factors: high

Within RNM, there is a well defined split between understanding what is happening in the system, and generating an effective plan to help alleviate the problem. In the former case, there are many real time data feeds from which knowledge about the system can be extracted, including loop detectors, ANPR (automatic number plate recognition), and CCTV. In the latter case, the traffic manager can manage a situation by initiating a range of actions; this includes the setting of traffic light timings, variable message signs (VMS), variable speed limits (VSL), ramp metering and radio broadcasting. In real time traffic control of large road networks it has been demonstrated that necessary processing and decision making is beyond the capabilities of human operators alone, and as the demand for road usage increases, this difficulty in managing traffic effectively becomes more acute. Additionally, the cost of congestion is increasing over time and in the UK alone is expected to rise to £30 billion by 2010. Improvement to the efficiency of traffic control and management also can be linked to the reduction of emissions of air pollutants produced by road traffic.

An application for AI planning could be to generate traffic and transport system plans and courses of actions in real-time to enable more effective control of incidents and events. A similar application might be to help with crisis management across the LA and HA controlled networks by generating plans which take account of LA and HA priorities and interactions. Hence there is a clear aim and motivation for the introduction of this kind of technology: to increase the quality of plans (which involve lights, VSLs, VSMs, etc), taking into account an increasing amount of information flow, which will benefit the quality of life through reduced congestion and polutant emissions.

Technological Context and Human Factors

Existing technological infrastructure: high

There has been a good record of adoption of computer systems in road network traffic management, and currently there are emerging common service platforms which will be beneficial to products and services delivered by technology providers. High level data platforms such as the HA's *Travel Information Highway* allow sophisticated software packages to both monitor and disseminate traffic information both to other services and to the general public (eg in the uk we have www.trafficengland.co.uk). The development of self-adapting computer systems such as SCOOT[1] has been one of the most important single developments. SCOOT systems are used worldwide to control the timings and offsets of groups of traffic lights connected by a local road network. They adapt to different traffic levels, automatically adjusting light timings at related junctions in reaction to sudden or gradual changes in traffic flows.

Data availability and quality: medium to high

In the UK, the UTMC[2] is a relational database convention for data collected and distributed in the course of traffic management. UTMC provides a high level, standard platform for traffic applications to use and interoperate. Local Authorities use systems such as SCOOT and UTMC to make effective and efficient use of technology in managing the local road network. However, there are some shortfalls with current systems, and rising traffic levels will only exacerbate the situation. The most serious problem is that, although motorways and city centres have traffic flow monitored, traffic flow outside of these areas is largely unknown, and there is still a high degree of uncertainty of the status of some networks.

Regarding evaluation platforms for testing new technology off-line, there is a long standing history of transport research using such methods, with large amounts of data available for testing and simulation.

Human Factors: high

Expertise in management and operation of the network appears to be thin, and there is a realisation within the service that this, and the growing complexity of the problem, will require more technological investment. There are a range of high-tech service providers in the sector who are experienced in technological innovation. All stakeholders appear ready to embrace further technological innovation (especially given the past success of SCOOT).

Knowledge Engineering Factors

Closeness to previous applications: medium

Within the area, there have been attempts to incorporate some kinds of specific automated reasoning systems into the control of motorway incidents eg in the MOLA system (Still, P.B and Harbord, B.J. 1998). No such attempt has been made in local authority-controlled roads in the UK.

Regarding similar domains, the Pipesworld domain from IPC-4 shares some characteristics with road transport: the basic domain consists of an arcs and nodes network, with some arcs (roads) bi and some uni-directional. Also, the 'transporter' (pipe or road) does not move - objects move along them. Despite there being ways to abstract the complexity of road networks (eg by bundling traffic into distinct quanta) the complexity of the road network may well cause a problem of scale to current planning engines.

Procedure formalisation in the problem area: medium

Plans do exist on paper, but are not plentiful. Decisions and plans are made by experts on the basis of collated information of the road network. Current procedure formulation is at the level of SQL constructs.

Appropriateness for AI planning solution: medium

Parameterised actions can be formed to model the actions mentioned above, although the effects of such actions may be difficult to encode in propositional form. Propositional descriptions of road network status and goal criteria are not generally used in current systems.

Application Area: Flood prevention and management

Flood prevention and management (FPM) involves, as in RNM, local and national authorities, service industries, and research institutes. This is due to its perceived importance: throughout many parts of the world the prevention, early warning, crisis and post-crisis management of water innundation is an important factor in human well-being. We have identified two areas which incorporate two potential applications of AI planning: for long term planning of infrastructure to prevent or lessen the risk of flooding, and for real-time planning to support flood event management. The former area considers such criteria as climatic change and population change, and may involve flood defence design or even river design. The latter area falls under the heading of crisis management, and may incorporate evacuation mangement. Here (as in RNM) there is the need to understand what the status of the event is - this is essential to support the active management of any identified problems.

Below we concentrate on evaluating the feasibility of AI planning to support flood event management, and use the information from deliverables of the current EU project 'FLOODsite' to support it [3] FLOODsite aims to develop tools to help in evacuation management, particularly meta-tools and frameworks for the building of specific decision support systems (DSS).

[1] http://www.scoot-utc.com/

[2] http://www.utmc.gov.uk/

[3] http://www.floodsite.net/

Evaluation using Criteria

Motivation Factors: high

The need for plan generation support in the real-time scenario is directly supported by FLOODsite research: 'Given the large variety of possible scenarios generating flash floods, the pre-flood generation of all the corresponding emergency plans is out of reach' (FLOODsite workplan, page 23). This implies that the motivation is similar to incident management in the RNM application - to be able to produce sound plans in real time in response to a crisis in which there are a number and mix of information streams.

Context and Human Factors

Existing technological infrastructure: medium - high

There are many decision support systems that have been created to help in flood event management in the UK, France and Netherlands alone (Rob 2007). These DSS are typically GIS-based simulation systems with user-friendly interfaces. They can inform on flood distributions, identify population, transport and properties at risk; evaluate the likely effectiveness of flood defences etc. Communication between 'actors' is very important in flood event management (as in other incident/crisis management) and hence systems are aimed at information dissemination among emergency services and connected organisations.

The number of decision support systems suggests a high level of technological infrastructure. However, real-time use of technology within the sector appears to be targeted at disseminating information about the unfolding crisis to the range of emergency services that are called upon to assist. No systems seem to exist that perform support for flood event management in general, or evacuation planning for flood events in particular, by generating plans in response to a specific disaster. Indeed, in the area of flood event management, we could find no evidence that there exists systems that can validate or simulate pre-existing evacuation plans; that is systems that input water distribution models, and simulate the execution of disaster plans in real time, and evaluate them.

Data availability and quality: medium - high

Data from meteorological predictions, data concerning population densities, population characteristics, physical assets (safe building etc) and evacuation routes is readily available. On the other hand, while obtaining data for simulation is possible, it is currently not possible (according to FLOODsite) to generate up to date models of water levels, velocities etc in real time, due to the amount of computational time required. Hence any simulation systems would need to use precomputed models.

Human Factors: medium - high

Research and innovation in this area is accepted as an essential ongoing activities by stakeholders in the field, hence there would be no threats to feasibility. Many of the potential users, however, would not be IT literate and hence any AI software would need to be embedded within user-friendly interfaces.

Knowledge Engineering Factors

Closeness to previous applications: medium

This area is clearly related to the more general area of crisis prevention and management. There has been a great deal of work on decision support for crisis or disaster management, ranging back more than 20 - 30 years, although only a fraction of this work has attempted to automate generation of plans. An exception is the ongoing work aimed at disaster management for eruptions of the Popocatepetl volcano in Mexico, where the techniques used are based on answer-set programming (Cortes, Solnon, & Martnez 2004). This work is aimed at integrating a planning function with existing GIS systems. The language used for representation incorporates some measures of uncertainty, and the system has the potential for generating simple emergency evacuation plans. However, the application appears as yet not implemented.

Evacuation planning is an activity that has already been used with the Planning community - it is used as an example within the recent textbook (Ghallab, Nau, & Traverso 2004). SIADEX (Fdez-Olivares *et al.* 2006) is a system that is currently undergoing tests in real fire fighting situations. It produces plans, monitors execution, and interacts with human experts to support management in forest fire fighting. The insights resulting from the SIADEX implementation would certainly contribute to the success of a flood event management application.

Procedure formalisation in the problem area: low - medium

In general, plans and procedures in the area are not formalised and if they exist are stated in natural language. However, there are some DSS that expect emergency response plans as an input, and evaluate them by calculating the effect. This implies the existence of some plan formulations.

Appropriateness for AI planning solution: medium

Many of the inputs required in a planning domain model have been formalised in past decision support systems: actions and methods representing resources to be used for evacuation, and objects such as carriers and routes (eg road networks). The planning state would likely consist of flood levels, safe evacuation zones, spatial distribution of inhabitants, types of inhabitants (eg able-bodied or not). While this appears appropriate for AI planning technology (and the closeness of this domain to general disaster management is evident) the continuous nature of the domain, in particular flood distribution, may be difficult to represent with current domain model languages.

Discussion

For the RNM application, strengths lie in the technological infrastructure, the motivation for the work, and (in the UK at least) the availability of interoperable services due in part to the standard technology platform (UTMC). The existence of software in the industry with AI characterists (SCOOT and MOLA) is an important factor. The main problems seem to be the lack of high level information about road network status (which equates to the 'world state' in planning), and the lack of precisely defined plan databases. For FPM, again, motivation and technological infrastructure and innovation is generally high. The main areas of concern are within the knowledge engineering aspects, particularly plan reasoning and representation aspects. In both areas then, it would seem that the applications are feasible, but more work is required to quantify the resources required to complete the knowledge engineering task.

Another application area we have investigated resulted in a remarkably different result in the feasibility criteria, leading to us not pursuing the application of AI planning. This is an historical example (in that it may not still hold today give the changes in technology) from the area of Air Traffic Control. It is based on our early work in formalisation of ATC separation criteria (McCluskey *et al.* 1995). The application area is to produce a planning aid for helping conflict resolution of aircraft during en-route control over North Atlantic airspace. Thus the planner would need to take existing route plans, adjust them to clear any airspace conflict that had been detected by a conflict probe, and output the new plans to an air traffic control officer. In this domain the level of current technology was high, data on aircraft positions and plans was very good, and offline evaluation was possible. Additionally, the knowledge engineering aspects were good: there were rule books, formalised plans, propositional state descriptions, and much of the context knowledge had been formalised through our previous work on aircraft separation criteria. The criteria that scored low were motivation and human factors: investigation showed that although automated aids were desired by some state holders, the en route air traffic control officers were quite happy with their current method, which was capable of delivering the plans without the need for extra technology.

Conclusions

In this paper we introduced a set of criteria for evaluating the feasibility of introducing planning technology into an application area. We applied these criteria to two application areas which (as yet) have not seen AI planning applications. Although with the applications considered the introduction of AI planning was thought to be feasible (with some reservations), the exercise appears to illustrate to us the difficulty in finding suitable application areas: an application must score well on all three aspects: motivation, technological infrastructure and knowledge engineering aspects.

Some aspects of the criteria were based on those that would be used when assessing an application for the introduction of a KBS, as the same problems of knowledge elicitation and availability of expertise may be evident. In control applications, however, a further important factor seems to be the level of technological progression within the industry. In order to integrate planning technology into a currently human controlled system, there should already exist high levels of technological use and expertise in the industry, such as examples of past success with AI technology. A parallel can be drawn with the field of Autonomic Computing (Kephart & Chess 2003), which is to do with the manufacture of omputer systems which take care of themselves in that they can self-configure, self maintain, self-heal etc. The protagonists of AC portray the deployment of autonomic qualities as the culmination of a technological progression along which the progress of an application area can be tracked. Hence, for an application area to adopt a new system incorporating autonomic features, the current technology must already be far advanced (eg the current system may have software components with intelligent characteristics). This seems the case with AI planning technology also: the application area in general must be technologically sophisticated enough to support knowledge engineering of the required dynamic and heuristic knowledge to make plan generation feasible.

Acknowledgements

I would like to thank personnel from the following organisations for their input: telent, AVLC Ltd, AeGis, TSEU, TRL, NASA, Wallingford Software, and Kent and Hampshire County Councils.

References

Biundo, S.; Aylett, R.; Beetz, M.; Borrajo, D.; Cesta, A.; Grant, T.; McCluskey, T.; Milani, A.; and Verfaillie, G. 2003. PLANET technological roadmap on AI planning and scheduling. Electronically avaliable at http://www.planet-noe.org/service/Resources/Roadmap/Roadmap2.pdf.

Cortes, C. Z.; Solnon, C.; and Martnez, D. S. 2004. Planning operation: An extension of a geographical information system. Proceedings of the 1st Intl. LA-NMR04 Workshop, Antiguo Colegio de San Ildefonso, Mexico City, D.F , Mexico.

Fdez-Olivares, J.; Castillo, L.; Garcia-Perez, O.; and Reins, F. P. 2006. Bringing users and planning technology together: experiences in SIADEX. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, 11 – 20.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann ISBN 1-55860-856-7.

Kephart, J. O., and Chess, D. M. 2003. The vision of autonomic computing. *Computer* 36(1):41–50.

McCluskey, T. L.; Porteous, J. M.; Naik, Y.; Taylor, C. N.; and Jones, S. 1995. A Requirements Capture Method and its use in an Air Traffic Control Application. *Software–Practice and Experience* 25:47–41.

Myers, K.; Frank, J.; McCluskey, L.; and Yorke-Smith, N. 2007. Workshop on Moving Planning and Scheduling Systems into the Real World. http://icaps07.icaps-conference.org/.

Rob, M. 2007. Review of flood event management decision support systems. Technical Report FLOODsite deliverable T19-07-01, Delft Hydraulics.

Still, P.B and Harbord, B.J. 1998. Strategic management of traffic in Kent using MOLA. In *9th International Conference on Road Transport Information and Control.*

Various. 1999. UMTC 04: Research Review and Requirements Report. Birmingham City Council and Leicester City Council and University of Leeds, http://www.utmc.gov.uk/research/.

Planning in Supply Chain Optimization Problem

N.H. Mohamed Radzi, Maria Fox and Derek Long
Department of Computer and Information Sciences
University of Strathclyde, UK

Abstract

The SCO planning problem is a tightly coupled planning and scheduling problem. We have identified some important features underlying this problem including the coordination between actions, maintaining temporal and numerical constraints and the optimization metric. These features have been modeled separately and experimented with the state-of-the-art planners, Crikey, Lpg-td and SgPlan$_5$. However, none of these planners are able to handle all features successfully. This indicates a new planning technology is required to solve the SCO planning problem. We intend to adopt Crikey as a basis of the new technology due to the capability of solving the tightly coupled planning and scheduling problem.

Introduction

The Supply Chain Optimization (SCO) covers decision making at every level and stage of a system that produces products for a customer. The foremost important issues include the decisions about the quantities of products to be produced, scheduling the production and delivery whilst minimizing utilization of resources by the system within a certain planning period. All these decisions require reasoning and planning: understanding the factors that are relevant to the decisions and evaluation of the combinatorial of the problem. This means the planning process in SCO is not only deciding which action should be chosen to reach the goal state based on the logical constraints but also what is the consequence of selecting the action to the given optimization function. Due to these features the planning problems in SCO are different from the standard planning problem. The SCO planning domains are richer in temporal structure than most temporal domains in standard planning.

Temporal domains were introduced in the third International Planning Competition (IPC3) along with the temporal planning language PDDL2.1 (Long & Fox 2003)(Fox & Long 2003). The durative action is introduced as a new feature in the language. This feature allows actions in domains to be allocated a unit of time specifying time taken to complete said action. (Weld 1994). Furthermore, the quality of the plan is also measured by the overall length or duration of the plan generated. The temporal features in the language were later extended by the introduction of timed initial literals in PDDL2.2 (Edelkamp & Hoffman 2003). This is the language used in IPC4. Timed initial literals provide a syntactically simple way of expressing the exogenous events that are both deterministic and unconditional (Cresswell & Coddington 2003). Another way to express exogenous events was then introduced in PDDL3.0 by using hard and soft constraints (Gerevini & Long 2006). Hard and soft constraints express that certain facts must be, or are preferred to be, true at a certain time point as benchmarked in IPC5 (Dimopoulos *et al.* 2006).

Temporal domains in IPC3 require certain facts to be true at the end of the planning period. Although domains with deadlines or exogenous events are modeled in IPC4 and 5, none of these domains require actions overlap in time. In contrast, SCO domains require some collections of facts to be true not only at a particular final state but also throughout the trajectory. For example, some quantities of a product may be required to be in production throughout the planning period. Add to that, the SCO problems also require that actions to be executed concurrently during the planning process. For instance, there are exogenous events such as order deadlines that have to be met. We have to maintain these deadlines and concurrently execute other production activities. Moreover, there might be some threshold values that have to be maintained over the planning period.

As well as temporal structure, SCO domains are also rich with numerical structure. The domains with numerical structure were also introduced in IPC3. But most of the competition domains in the IPCs mainly deal with the consumption of resources and cost. In the SCO problems, numerical facts and constraints are used to model beyond the consumption of resources and cost. The numerical facts and constraints are also used to model the multiple actions: actions that have equivalent chances of being selected but the difference between them lies in the cost associated with performing them. In sum, SCO problems are very complex planning problems where temporal and numerical constraints enforced over time must be met as well as the logical constraints.

From another point of view, SCO planning problem is different from standard planning problems in terms of the way plans are constructed. The standard or classical planning problems concentrates on a process to find *what* actions should be carried out in a constructed plan by reasoning about the consequence of acting in order to choose among a set of possible courses of action (Dean & Kambhampati 1997). The number of actions required in a plan is usually unknown. The temporal planning problem is basically a combination of classical planning and scheduling. In the pure scheduling process, the actions are usually known and the choice of actions is limited compared to planning (Smith, Frank, & Jonsson 2000). The scheduling process concentrates on figuring out *when* and *with what* resources to carry out so as satisfy various types of constraint on the order in which the actions need to be performed (Dean & Kambhampati 1997). Therefore in temporal planning, the process of constructing a plan combines the decisions on *what* actions should be applied, with *when* it should be applied and *with what* resources (Halsey 2004).

The SCO problem however, is an example of a combinatorial problem that has string planning, scheduling and constraint reasoning components. Besides *what* and *when* choices it also contain choices about *how* to act. One way to introduce a *how* choice is to differentiate actions for achieving the same effect by numerical values such as duration or resource consumption. The *what* choices concern what resources are required for an action to be applied and the *when* choices concern how the action should be scheduled in to the rest of the plan. A very good example of the problem is the following: a manufacturer receives several orders from customers that consist of producing various quantities of several different items. These orders should be delivered within specified deadlines. The manufacturer has to schedule the production of each item. Due to the capacity limitations of the producer, the manufacturer has to decide which items should be produced using his own facilities and which items should be produced using other production options that are available. No matter how, the deadlines have to be met and the overall production cost should be minimized. In this case, the solution is not as simple as performing a sequence of actions but could involve executing many actions concurrently.

We have discovered that, although there are a number of planners in the literature that are capable of handling the individual features of PDDL2.1 and PDDL3, there are no planners currently available that can reliably solve non-trivial problems.

The reminder of this paper is structured as follows. First we present a description of a simple domain within the class of problems. We have encoded the domain and applied several state-of-the-art planners to it. The outcomes of the experiment are discussed in the following section but, in brief, the best performing planners in IPC4 and IPC5 are unable to solve the problems we set. Clearly, SCO problems encompass a huge variety and would in general be beyond the reach of any automated planner. Therefore, this discussion is followed by the definition of a subclass of problems that we intend to focus on in our work. We will develop a planner (by enhancing an existing planning system) that is capable of solving this subclass of problems. Later, we briefly describe our future work including the planner that we intend to enhance.

Domain Definition

A simple example of production planning problem in the supply chain is illustrated in Figure 1. The process starts with receiving the customers orders. Each order has a different combination of products and also different delivery deadlines. The process is then followed by selecting the production types of each product. In our example, each production type has a different processing time and cost: *normal-time*, *over-time* and *outsource*. The outsource action furthermore can be performed by several suppliers where each supplier is associated with a different lead time and cost. The domain demonstrates the properties discussed in the above section. The choices of production action represent the multiple choice of actions for achieving the same task. These actions can be executed simultaneously as well in parallel with other activities. The probability of the action being selected is dependent on the objective function. Any plan produced by the planner should minimize the overall cost and time taken to produce all items as well as meeting the specified deadlines. For example, item_1 can be produced either by the *normal-time* action or the *outsource* action but, choosing the *normal-time* action might cause a delay in the product delivery so that it is better to choose the *outsource* action. In an efficient plan we might be producing item_2 while we are also producing item_1. This domain has been encoded and presented to some of state-of-the-art planners.

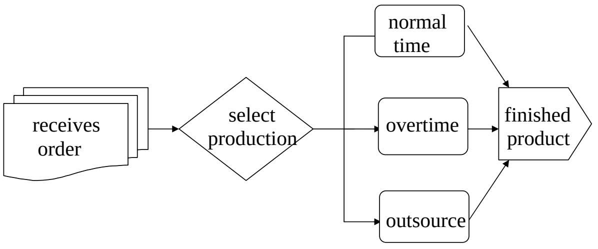


Figure 1: A Simple production process

State-of-the-Art Planners

We chose three different types of temporal planners for our experiments. All of these planners are claimed to be able to handle the temporal features of PDDL2.1 and also features to express deadline such as time windows and hard constraint. The planners are as follows:

SGPlan$_5$ is a temporal planner that received a prize for overall best performance in IPC5. The planner works with PDDL3.0, which features timed initial literals, hard constraints and preferences. It generates a plan by partitioning the planning problem into subproblems and finds a feasible plan for each sub-goal. The multi-value domain formulation (MDF) is used as a heuristic technique in the planner for resolving goal preferences and trajectory and temporal constraints (Chih-Wei *et al.* 2006).

The **LPG-td** planner is an extension of LPG (Gerevini & Serina 2002) that can handle features in PDDL2.1 and most features in PDDL2.2, including timed initial literals and derived predicates. The timed initial literals represent facts that become true or false at certain time points, independently of the actions in the plan. This feature can be used to model a range of temporal constraints including deadlines. LPG-td is an incremental planner that generates a plan in the domain involving maximization or minimization of complex plan metrics. An incremental process improves the plan by using the first generated plan to initialize a new search for a second plan of better quality and so on. It can be stopped at any time to give the best plan computed so far (Gerevini, Saetti, & Serina 2004).

CRIKEY is a temporal planner that solves a tightly coupled type of planning and scheduling problem. This planner supports PDDL2.1. It has implemented the envelope and content concept in order to handle the communication between the planning and scheduling. Content actions are executed within envelope actions. Therefore the minimum length of time for the content actions must be less than or equal to the maximum total length of time for the envelope actions (Halsey, Long, & Fox 2004). The envelope and content concepts were introduced to allow Crikey to solve problems in which actions must be executed in parallel in order to meet temporal and resource constraints.

Experimental Results

The aim of the experiments is to investigate the capability of each planner to cope with the following features:

(1) temporal constraints that require facts to be maintained over time; (2) optimization metrics including temporal and numerical optimization; (3) coordination and concurrent actions.

The domain described in the previous section was encoded using PDDL. There were six actions modelled in the domain including STACK_ORDER, CHOOSE_BRAND, OVERTIME, NORMAL_TIME, OUTSOURCE and SHIP_ON_TIME. Since different planners can work with different versions of PDDL, we have exploited PDDL2.1 features to represent domains presented to Crikey, PDDL2.2 for domains presented to Lpg-td and PDDL3.0 for domains presented to SgPlan$_5$. We have had to use different syntax to express the deadlines: timed initial literals for Lpg-td and hard constraints for SgPlan$_5$. No specific syntax is given in PDDL2.1 for expressing deadlines, but it is possible to encode them using envelope actions and clips (Fox, Long, & Halsey 2004; Cresswell & Coddington 2003). In the first experiment we have encoded only a single deadline. The encoded problem has been presented three times to each planner, each time with a different set-up. The problem instances are described in Table 1. For example in the first instance, the duration for actions NORMAL_TIME and OVERTIME are 7 and 8 unit time respectively. The OUTSOURCE action can be performed through either by *supplier*$_1$ or *supplier*$_2$ with the duration are 5 and 6 unit time respectively. The planners are expected to perform one of these actions in order to accommodate the deadlines. The duration of other actions defined in the domain is 1 unit time. Table 2 describes the deadlines and the plan duration given by each planner (if any) together with the action selected in the plan.

prob	normal-time	overtime	supplier$_1$	supplier$_2$
1	7	8	5	6
2	7	5	7	6
3	5	8	7	9

Table 1: problem instances set-up

prob	*d*	Crikey	SgPlan	Lpg-td
1	8.05	7.05 supplier$_1$	8.004 supplier$_1$	8.000 supplier$_1$
2	8.05	8.05 supplier$_2$	no solution	8.00 overtime
3	8.05	7.05 normal-time	no solution	8.00 normal-time

d: deadline

Table 2: maintaining time constraints by each planners

As we can see in Table 2, Crikey and Lpg-td planners perform very well in maintaining the temporal constraint. Both planners managed to obtain a plan with the most appropriate actions so that the completion time is within the deadline. But, SgPlan$_5$ only generates a plan for the first instance. There are no solutions for the second and third instances.

Later, the second experiments were carried out to see whether these planners can reason about the optimization metric, for example, minimize the makespan. For this purpose, the deadlines were excluded from the domains and then replaced with the minimization metric of *total-time*. The same encoded problems were applied to all planners. The description of the problem instances were remained the same as in the Table 1. Table 3 exhibits the completion time of the plan generated by each planner. We can see from this Table, Lpg-td was capable of minimizing the makespan compared to both SgPlan$_5$ and Crikey. SgPlan$_5$ as described in the Table 3 always choose the same action no matter the changes made in the duration of the actions in the domain. Therefore in experiment 1, SgPlan$_5$ unable to produce any plan for instance$_2$ and instance$_3$ since the set up time for the particular action has violated the deadlines. Crikey also performs similar to SgPlan$_5$ in this experiment. As depicted in Table 3, NORMAL_TIME action is chosen regardless the duration set up for the action in each instance. The result from experiment 1 also indicates that Crikey will only maintain the temporal constraint by finding a feasible solution but not an optimal solution. Refer to Table 2 for instance$_2$. Although plan duration given by Crikey meeting the deadlines, but the duration is slightly bigger than plan duration generated by Lpg-td.

prob	Crikey	SgPlan	Lpg-td
1	9.004 normal-time	8.004 supplier$_1$	8.000 supplier$_1$
2	9.004 normal-time	10.004 supplier$_1$	8.000 overtime
3	7.004 normal-time	10.004 supplier$_1$	8.000 normal-time

Table 3: optimization metric: minimizing makespan

Besides minimizing the makespan, some experiments to investigate whether these planners can reason about numerical values by giving a plan that minimizes the total cost incurred due to action selection were carried out. The problems used in experiment 2 were applied in this experiment. But, the optimization metric was changed to minimize *total-cost* and the same durations were set to each action. The number of instances were also increased to ten, each instance has been set up with a different cost. The metric value of the plan is given in the generated plan for the plan produced by SgPlan$_5$ or Lpg-td. But for Crikey the metric value can be identified through the action selected in the plan. Table 4 shows the metric values or total cost obtained from the plan generated by each planner. Lpg-td produced plans that minimized the total cost for every instances. In some instances, either SgPlan$_5$ or Crikey also able to produce the optimized plans. The optimized plans were obtained due to the cost of the actions that are considered to be selected have the smallest cost compared to other actions in the problem. This is definitely not because of the capability of the planner to reason on the numerical values. The domains and problems involved in the experiment can be accessed at http://www.cis.strath.ac.uk/∼nor.

problem	Crikey	SgPlan	Lpg-td
1	9.00	11.00	3.20
2	7.00	4.00	3.20
3	9.00	11.00	4.20
4	12.00	11.00	7.20
5	12.00	5.00	5.20
6	4.00	5.00	4.20
7	7.00	12.00	7.20
8	20.00	15.00	13.20
9	20.00	9.00	9.20
10	4.00	9.00	4.20

Table 4: optimization metric: minimizing total-cost

As mentioned in the previous section, the SCO contains choices about how to act. The choices of how to act affect the quality of solution as well as satisfiability of the schedule. We cannot simply perform the action selection first and later schedule the actions according to their temporal and numerical information. This means the planning and scheduling tasks are tightly coupled and cannot be performed separately. This situation requires coordination between actions and execution of the concurrent actions. Coordination is where the actions can happen together and interact with another. Meanwhile concurrency means more than one action happen simultaneously but they are not to interfere with each other (Halsey 2004). For example see Figure 2. There are three deadlines, denoted by x_1, x_2 and x_3. The x_2 and x_3 happen at the same time point. These deadlines x_1, x_2 and x_3 require actions (a_1,a_2,a_3), (a_1,a_4,a_5) and (a_1,a_4,a_6) respectively. Either some parts or all parts of the actions' durations are overlapped in time or executed in parallel. The actions a_2 and a_3 must interact with action a_1. These actions must execute during the life time of action a_1. But there is no interaction between a_2 and a_3. The actions are required to execute simultaneously in order to achieve the deadline. The actions a_4, a_5 and a_6 are also examples of coordination where action a_5 and a_6 are executed in some portion of the life time of a_4. Furthermore, the a_5 and a_6 actions demonstrate the choice of how to act. In achieving deadlines x_2 and x_3, either a_5 or a_6 has to be executed following a_4. Another clear example of a domain in which some actions must happen in parallel, which has been investigated in the previous literature, is the Match Domain (Halsey, Long, & Fox 2004). However, the choices on how to act is not demonstrated in this domain.

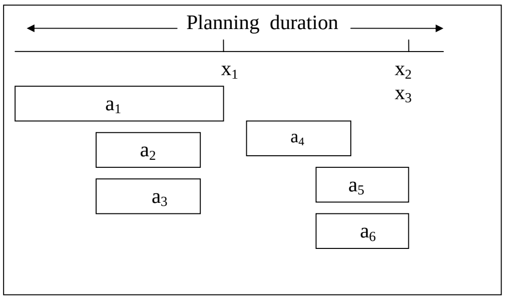


Figure 2: concurrent actions

Due to the importance of the above features in the SCO domain, we also investigate the capability of these planners to support these requirements. The coordination and concurrent features were indirectly performed in the temporal constraint problem to which Crikey was applied in experiment 1. In Crikey, actions are either *wrappers* or *contents* with wrappers containing contents and contents being completely contained within wrappers. Some content actions are also wrappers for other actions. In experiment 1, we have encoded deadlines as the wrapper actions. The other six actions decribed in the beginning of this section were the content actions. These content actions have to start after the wrapper start and end before the wrapper end. In other words, the wrapper and the content actions are performed in parallel. The wrapper and the content actions can be illustrated as actions (a_1,a_2,a_3) in Figure 2. Lpg-td and SgPlan$_5$ were then applied to the same domain, since both planners are capable of handing all PDDL2.1 features. Unfortunately, neither planner can solve the problem. As in Table 5, besides SCO domain, two other domains including the match domain were also tried. The driverlog-shift domain (Halsey 2004) is an extension of the driverlog domain used in IPC3. In this extended domain, the driver can only work for a certain amount of time or in a shift. The shift action is modelled as an envelope action. Therefore, driving and walking actions must be fitted into the shift action. SgPlan$_5$ produced a plan for this domain. But, in the plan, the walking and driving actions are performed after the shift action finished. In other words, they are performed in a sequence. SgPlan$_5$ and Lpg-td are able to perform concurrent actions, provided that the actions do not interfere with each other. This is as a result of both planners generating the temporal planning problem by finding out the sequential solution first and rescheduling them using temporal information.

domain	Crikey	SgPlan	Lpg-td
SCO	plan obtained	no solution	no solution
match	plan obtained	no solution	no solution
driverlog-shift	plan obtained	plan obtained	no solution

Table 5: domain with concurrent actions

Moreover, domains that are encoded with coordination or concurrent actions will have plans that shorten the makespan. Refer to Table 2. Although Crikey and Lpg-td choose the same action, the plan duration generated by each of the planner is different. The plan produced by Crikey has a shorter duration than the plan generated by Lpg-td.

The overall performance based on the criteria outlined or properties underlying in the SCO problems in the experiment are summarized in Table 6. Crikey is very good at maintaining constraints and coordination of tasks but very poor at metric optimization. Nevertheless for this problem, Crikey is still able to produce a feasible plan. Lpg-td, although it has a very good performance both in maintaining constraints and optimization, cannot perform coordination of actions. When this is required no plan can be produced at all. Although, SgPlan$_5$ can handle temporal constraints as benchmarked in IPC5, the domains involved do not include choices about how to act. An example arises in the truck domain. This domain only encodes what action should be carried out in order to meet the temporal constraints. SgPlan$_5$ seems unable to reason with choices about how to act. Therefore for some instances in experiment conducted, SgPlan$_5$ did not produce any plan. Unlike Lpg-td, SgPlan$_5$ is sometimes able to produce a plan for a concurrent domain but the execution of actions in the plan are performed in a sequenced manner.

planner	time constraint	optimization metric	coordination
Crikey	very good	poor	very good
Lpg-td	very good	very good	cannot performed
SgPlan	poor	poor	cannot performed

Table 6: overall performance of planners

Subclass of SCO problem

As discussed in the beginning of the paper, SCO is a hard combinatorial problem that requires not only reasoning about the logical relations between actions but also has to examine the temporal and numeric relations between actions. Since it is very hard to solve the overall problem features, only the subclass of this problem will be focused on in this research. The properties of the subclass problem are identified as follows. The very important properties are maintaining temporal and numerical constraints. The second feature is the optimization metric in term of numerical values. All these properties require coordination between actions as well as actions to be performed concurrently in the generated plan. Since planning problems have a strong scheduling element, we will have a selection of alternative actions (planning) within the large selection of actions described in the domain. This situation exhibits the how choices action in the domain. Within the alternative actions, there is also a selection of possible resources, giving rise to a scheduling problem. All these actions are weighted by numerical values representing their costs. At this stage we are not interested in optimization in term of temporal metrics.

Conclusion

This paper discusses the features of SCO planning problems and investigates the performance of state-of-the-art planners on domains with these features. We have run the experiments on the individual features separately. The planners are expected to handle some of the features, such as minimization of *total-cost* or *total-time* metric as well as satisfying the hard constraints. However, as we can see, none of the state of the art planners we tried were able to successfully handle all the features. Therefore, experiments conducted to date have identified several improvements in the planning technology that are required in order to solve the SCO type of domain.

Future Work

In the near future we will develop a subclass of the SCO problem that combines all the features together. The more complex optimization metric will be included in the problem since the numerical features considered in the experiment so far are very simple. As numerical constraints are identified as one of the properties of the SCO subclass, the numerical constraints will also be included in the domain. The domain will be used to test a variety of planners. We plan to adopt Crikey as the basis of the new technology that we intend to develop. Crikey is chosen due to its ability to cleanly manage the tightly coupled interaction between planning and scheduling as well as other features such as duration inequalities and interesting metric optimisation.

References

Chih-Wei; Wah, B.; Ruoyun; and Chen, Y. 2006. Handling soft constraints and goal preferences in SGPLAN. In *ICAPS Workshop on Preferences and Soft Constraints in Planning.* ICAP.

Cresswell, S., and Coddington, A. 2003. Planning with timed literals and deadlines. In Porteous, J., ed., *Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, 22–35. University of Strathclyde. ISSN 1368-5708.

Dean, T., and Kambhampati, S. 1997. Planning and scheduling. In *The Computer Science and Engineering Handbook 1997.* CRC Press. 614–636.

Dimopoulos, Y.; Gerevini, A.; Haslum, P.; and Saetti, A. 2006. The benchmark domains of the deterministic part of IPC-5. In *Booklet of the 2006 Planning Competition, ICAPS'06.*

Edelkamp, S., and Hoffman, J. 2003. PDDL2.2: The language for the classical part of the 4th international planning competition.

Fox, M., and Long, D. 2003. An extension of PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Fox, M.; Long, D.; and Halsey, K. 2004. An investigation into the expressive power of PDDL2.1. In *Proceedings of ECAI'04.*

Gerevini, A., and Long, D. 2006. Plan constraints and preferences in PDDL3. In *ICAPS Workshop on Preferences and Soft Constraints in Planning.* ICAPS.

Gerevini, A., and Serina, I. 2002. LPG: A planner based on local search for planning graphs. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling.*

Gerevini, A.; Saetti, A.; and Serina, I. 2004. Planning with numerical expressions in LPG. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-04).* IOS-Press, Valencia, Spain.

Halsey, K.; Long, D.; and Fox, M. 2004. CRIKEY - a planner looking at the integration of scheduling and planning. In *Proceedings of the Workshop on Integration Scheduling Into Planning at 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, 46–52.

Halsey, K. 2004. *CRIKEY! Its Co-ordination in Temporal Planning: Minimising Essential Planner–Scheduler Communication in Temporal Planning.* Ph.D. Dissertation, Ph. D. Dissertation, University of Durham.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research* 20:1–59.

Smith, D. E.; Frank, J.; and Jonsson, A. 2000. Bridging the gap between planning and scheduling. *The Knowledge Engineering Review* 15:47–83.

Weld, D. S. 1994. An introduction to least commitment planning. *AI Magazine* 4.

Velocity Tuning in Currents Using Constraint Logic Programming

Michaël Soulignac[*,**] **Patrick Taillibert**[*] **Michel Rueher**[**]

* THALES Aerospace
2 Avenue Gay Lussac
78852 Elancourt, FRANCE
{firstname.lastname}@fr.thalesgroup.com

** Nice Sophia Antipolis University
I3S/CNRS, BP 145
06903 Sophia Antipolis, FRANCE
rueher@essi.fr

Abstract

Because of its NP-hardness, motion planning among moving obstacles is commonly divided into two tasks: path planning and velocity tuning. The corresponding algorithms are very efficient but ignore weather conditions, in particular the presence of currents. However, when vehicles are small or slow, the impact of currents becomes significant and cannot be neglected. Path planning techniques have been adapted to handle currents, but it is not the case of velocity tuning. That is why we propose here a new approach, based on Constraint Logic Programming (CLP). We show that the use of CLP is both computationally efficient and flexible. It allows to easily integrate additional constraints, especially time-varying currents.

Introduction

Mobile robots are more and more used to collect data in hostile or hardly accessible areas. For physical or strategic reasons, these robots may not be able to receive directly orders from a headquarter in real-time. Thus, they have to embed their own motion planner. Because the environment is often changing or unknown, this planner has to be very reactive.

Motion planning is yet a complex task, answering to two questions simultaneously: where should the robot be, and when? It is known to be a NP-hard problem (Canny 1988). That is to say, the computation time grows exponentially with the number of obstacles.

To guarantee a reasonable response time, motion planning is commonly divided into two simpler tasks: (1) a *path planning* task, dealing with the question *where*, and (2) a *velocity tuning* task, dealing with *when*.

Algorithms associated to these two tasks are generally based on simple assumptions. For instance, obstacles are often modeled as polygonal-shaped entities, moving at constant velocity. Data about weather, in particular about (air or water) currents, are usually ignored.

However, in the case of Unmanned Air Vehicles (UAVs) or Autonomous Underwater Vehicles (AUVs), which may be small or slow, the impact of currents is significant. So, ignoring currents can lead to incorrect or incomplete planners. Such planners may return a physically infeasible path, or no path at all, even if a valid path exists.

Some extensions have been developed in the field of path planning, but currents remain neglected during velocity tuning.

That is why we propose here a new velocity tuning approach, based on Constraint Logic Programming (CLP). Our experimental results show that this approach is computationally efficient. Moreover, it offers a flexible framework, allowing to easily integrate other constraints, such as time-varying currents or temporal constraints.

This paper is organized as follows. Section I recalls the existing planning methods. Section II formalizes the problem of velocity tuning in presence of currents. Section III introduces our modeling of this problem in terms of a Constraint Satisfaction Problem (CSP) on finite domains. Section IV proposes examples of additional constraints. Finally, section V provides some experimental results, obtained on real wind charts.

I. Motion planning in currents

The decomposition of motion planning into path planning and velocity tuning tasks was first introduced in (Kant & Zucker 1986). This decomposition is widely used in robotics because both tasks can be done in a polynomial time.

However, it has to be noticed that it is source of incompleteness: the path planning phase may generate a path which is unsolvable in the velocity tuning phase.

1. Path planning

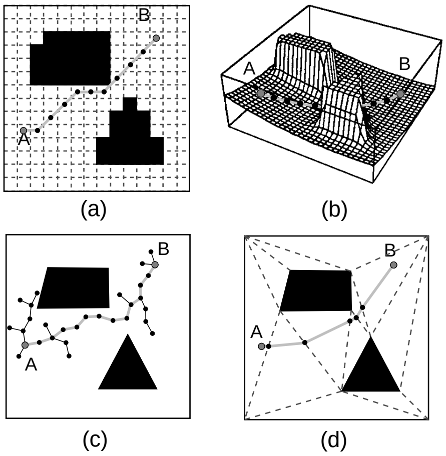
Path planning methods consist in finding a curve between a start point A and a goal point B, avoiding static obstacles O^i (generally polygonal-shaped). They can be divided into four categories: (1) decomposition methods, (2) potential fields methods, (3) probabilistic methods, and (4) metaheuristics.

Graph decomposition methods (fig. 1a) are based on a discretization of the environment into elementary entities (generally cells or line segments). These entities (plus A and B) are then modeled as nodes of a graph G. The initial -i.e. concrete- path planning problem is thus reformulated into an abstract one: find the shortest path from node A to node B in G. To do this, classical search techniques are applied, such as the well-known A^* algorithm (Nilsson 1969) or one of its numerous variants.

Potential field methods (fig. 1b) (Khatib 1986) consider the robot as a particle under the influence of a potential field U, obtained by adding two types of elementary fields: (a) an attractive field U_{att}, associated to B and (b) repulsive fields U^i_{rep}, associated to obstacles O^i. The point B corresponds to the global minimum of the function U. The path between A and B can thus be computed by applying gradient descent techniques in U values, starting from A.

Probabilistic methods (fig. 1c) (LaValle 1998) are based on a random sampling of the environment. These methods are a particular case of decomposition methods: random samples are used as elementary entities, linked to their close neighbors, and modeled by a graph. Probabilistic RoadMap (PRM) and Rapid Random Trees (RRT) are the most famous methods in this category.

Metaheuristics refer to a class of algorithms which simulate natural processes (fig. 1d) (Zhao & Yan 2005). The three main metaheuristics applied to path planning are: (a) genetic algorithms, inspired by the theory of evolution proposed by Darwin; (b) particle swarm optimization, inspired by social relationships of bird flocking or fish schooling; (c) ant colony optimization, inspired by the behavior of ants in finding paths from the colony to food.



(a) (b)

(c) (d)

Figure 1: Paths (in light grey) obtained by the following methods: (a) A^* algorithm on regular cells; (b) potential fields; (c) RRT; (d) particle swarm optimization.

All these methods have two common characteristics: (1) the cost $\tau(M, N)$ between two points M and N represents the Euclidean distance $d(M, N)$ and (2) the computed path is made up of successive line segments. This last property is the base of our modeling, described in section II.

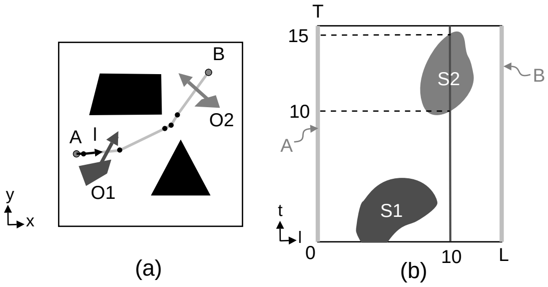
However, in presence of currents, the fastest path is not necessary the shortest. To illustrate, let us consider a swirl: the fastest way to link A and B is more circle-shaped than linear.

In this context, new cost functions have been proposed, to make a compromise between following the currents and minimizing the traveled distance (Garau, Alvarez, & Oliver 2005)(Petres *et al.* 2007).

2. Velocity tuning

The existing velocity tuning approaches generally work in a 2-D space-time. The first dimension $l \in [0, L]$ (where L is the length of the path) represents the curvilinear abscissa on the path. The second one, $t \in [0, T]$ (where T is the maximal arrival time), the elapsed time since departure. In this space-time:

- Each point of the path is represented by a column. In particular, start and goal points are represented by the extreme left and right columns.
- Each moving obstacle O^i generates a set of *forbidden surfaces* S^i (often only one). These surfaces contains all couples (l, t) leading to a collision between the robot and O^i. For instance, in figure 2b, the abscissa $l = 10$ is forbidden between $t = 10$ and $t = 15$.



(a) (b)

Figure 2: (a) path of fig. 1d, adding two moving obstacles; (b) the corresponding 2-D space-time.

Once the space-time is built, the initial velocity tuning problem can be reformulated into a path planning problem in this space-time. However, this space-time has specific constraints, notably due to time monotony or velocity bounds. Therefore, specific methods have been applied, like: (1) adapted decomposition methods, (2) B-spline optimization, and (3) the broken lines algorithm.

As explained before, decomposition methods (figure 3a) divide the space-time into elementary entities and apply graph search techniques. Since a lot of paths are temporally equivalent (they arrive at the same time), an appropriate cost is necessary. For instance, (Ju, Liu, & Hwang 2002) used a composite cost function balancing the arrival time and the velocity variations.

B-spline optimization techniques (figure 3b) consist in representing the optimal trajectory in the space-time by a B-spline function (Borrow 1988), parameterized by some control points K^i. Graphically, the points K^i locally attracts the curve of the B-spline. Their position is computed in order to minimize the mean travel time, using interior point techniques.

The broken lines algorithm (figure 3c) (Soulignac & Taillibert 2006) tries to link A and B using a unique velocity, i.e. a unique line segment in the space-time. At each intersection of the line with a surface S^i, a velocity variation is introduced, by "breaking" this line into two parts. To sum up, this algorithm tries first to arrive as earlier as possible, and then to minimize velocity variations.

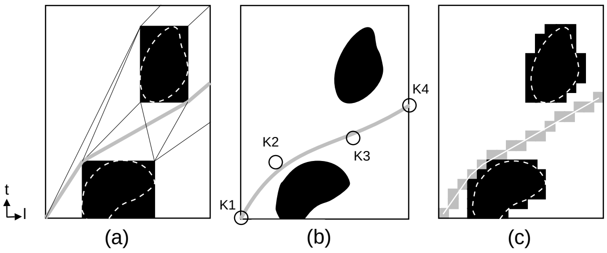


Figure 3: Paths (in light grey) obtained in the space-time of fig. 2 by the following methods: (a) visibility graph; (b) B-spline optimization with 4 control points; (c) broken lines algorithm.

All these methods neglect the influence of currents. This is acceptable in presence of weak currents, since trajectory tracking techniques such as (Park, Deyst, & How 2004) remain applicable to dynamically adjust the robot's velocity.

However, when currents become strong, the robot is neither guaranteed to stay on its path, nor to respect the time line computed by the velocity tuning algorithm. That is why we propose a new approach, based on CLP techniques.

II. Problem Statement

1. Informal description

A punctual robot is moving on a pre-computed path $\mathcal{P}$ from a start site A to a goal site B, in a planar environment containing moving obstacles and currents, with a bounded velocity.

It has to minimize its arrival time at B, with respect to the following constraints: (1) obstacle avoidance and (2) currents handling. Data about obstacles and currents are known in advance.

2. Formalization

The environment is modeled by a 2-D Euclidean space E, with a frame of reference $R = (0, x, y)$. In R, the coordinates of a vector $\vec{u}$ are denoted (u_x, u_y) and its modulus u.

The path $\mathcal{P}$ is defined by a list V of n viapoints, denoted V^i. Each viapoint V^i is situated on $\mathcal{P}$ at curvilinear abscissa l^i. Two successive viapoints (V^i and V^{i+1}) are linked by a line segment. In other terms, $\mathcal{P}$ is made up of successive line segments, which is the result of all path planning methods presented before.

Note that $\mathcal{P}$ is obtained by using adapted cost functions, incorporating the influence of currents (otherwise the velocity tuning would be meaningless).

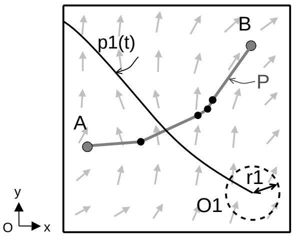


Figure 4: A velocity tuning problem with currents.

Each moving obstacle O^i is a disk of radius r^i. This disk corresponds to a punctual mobile surrounded by a circular safety zone. The position of the mobile -i.e. the center of the disk- is given at every time t by p^i. Note that contrary to most approaches, there is no restriction on the function p^i.

Finally, the current can be seen as a 2-D vector field $\overrightarrow{c}$, known either by measurement or forecasting. Thus, the data about $\overrightarrow{c}$ are, by nature, discontinuous, i.e. defined on the nodes of a mesh (not necessary regular), called *current nodes*. The mean distance between current nodes may correspond to the resolution of measures or the precision of the forecast model.

The robot's velocity vector relative to the frame R (ground speed) is denoted $\overrightarrow{v}$, and its velocity vector relative to the current $\overrightarrow{c}$ (current speed) is denoted $\overrightarrow{w}$.

It is important to understand that $\overrightarrow{w}$ only depends on the engine command, whereas $\overrightarrow{v}$ is impacted by the current $\overrightarrow{c}$. Indeed, applying the velocity composition law, the quantities $\overrightarrow{v}$, $\overrightarrow{c}$ and $\overrightarrow{w}$ are linked by the following relation:

$$\overrightarrow{v} = \overrightarrow{w} + \overrightarrow{c} \tag{1}$$

Our problem consists in finding a timing function σ:

$$\sigma : M \in \mathcal{P} \mapsto t \in [0, T] \tag{2}$$

minimizing the arrival time $t_B = \sigma(B)$, with respect to the following constraints:

1. maximal velocity: the modulus of the robot's velocity relative to the current, denoted w, is smaller than w_{max}. Note that the bound w_{max} only depends on the robot's engine capabilities;
2. obstacles avoidance: the robot has to avoid a set of m moving obstacles;
3. currents handling: the robot has to take into account disturbances due to the field $\vec{c}$.

The quantity T is called *time horizon*. It materializes the maximal arrival date to B. This upper bound may be due to the embedded energy or visibility conditions.

III. Velocity tuning using CLP

Velocity tuning using CLP consists in two steps: (1) defining the constraints describing the velocity tuning problem and (2) solving the corresponding CSP, with the adequate search strategies.

1. Data representation

The constraints above are defined on finite domains. Therefore, the initial data about the environment are reformulated using an appropriate representation.

Time representation

The interval $[0, T]$ is discretized using a constant step ε. The value ε depends on the context. In our applications, $[0, T]$ contains less than 1000 time steps. For instance, $T = 2$ hours and $\varepsilon = 10$ seconds leads to 720 time steps.

Currents representation

As we explained before, the current is known in a finite number of points, called current nodes, obtained by measurement or forecasting. Since current nodes already include an error, we think that it is meaningless to finely interpolate the value of the current between these nodes.

Therefore, we propose the concept of *Elementary Current Area* (ECA). An ECA is an polygonal region of the environment, in which the current is homogeneous. Each ECA contains a unique current node. The value of this node is extended to the whole area.

ECAs are computed by building the Voronoï diagram (Fortune 1986) around the current nodes. This diagram is made up of line segments which are equidistant to the nodes. It is illustrated in figure 5, for uniform and non-uniform distributions.

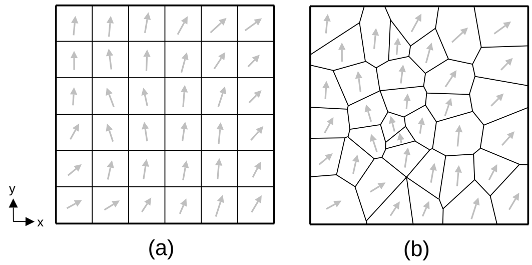


Figure 5: Illustration of ECAs for two distributions of current nodes (grey arrows): (a) uniform and (b) non-uniform.

Artificial viapoints

Artificial viapoints are additional viapoints guaranteeing that the current is constant between two successive viapoints. They are obtained by intersecting the path $\mathcal{P}$ and the borders of ECAs. Since both $\mathcal{P}$ and borders are made up of line segments, these intersections can be computed easily.

The initial list V of viapoints is thus enlarged into V', containing $n' > n$ elements. The current between two successive viapoints V^i and V^{i+1} is denoted $\overrightarrow{c^i}$.

2. Constraints definition

In this part, we show how the velocity tuning problem can be described thanks to two types of constraints: (a) constraints related to currents and (b) constraints related to moving obstacles avoidance.

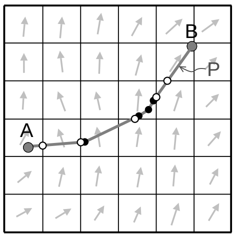


Figure 6: Artificial viapoints (white dots) obtained for the Voronoï diagram of fig. 5a. These viapoints are added to the initial viapoints (black dots).

Note that the currents are constant in time here (time-varying currents are considered in section IV).

a. Constraints related to currents

Let us consider the straight line move $\overrightarrow{d^i}$, between the viapoints $V^i = (x^i, y^i)$ and $V^{i+1} = (x^{i+1}, y^{i+1})$. For this move, we define:

- $\overrightarrow{c^i}$ the velocity of the current
- $\overrightarrow{v^i}$ the robot's velocity relative to the frame R
- $\overrightarrow{w^i}$ the robot's velocity relative to $\overrightarrow{c^i}$

As explained in equation 1, $\overrightarrow{v^i}$ and $\overrightarrow{w^i}$ are linked by $\overrightarrow{v^i} = \overrightarrow{w^i} + \overrightarrow{c^i}$. Moreover, since we want to impose the move $\overrightarrow{d^i}$ to the robot, $\overrightarrow{v^i}$ and $\overrightarrow{d^i}$ are collinear.

Thus, if we denote C^i the result of translating V^i by vector $\overrightarrow{c^i}$, we can build the vector $\overrightarrow{v^i}$ by intersecting:

- The line $\mathcal{L}^i$, of direction vector $\overrightarrow{d^i}$
- The circle $\mathcal{C}^i$, of center C^i and radius w^i

If I_1^i and I_2^i are the intersections obtained[1] (possibly confounded), $\overrightarrow{v^i}$ can be either the vector $\overrightarrow{v_1^i} = \overrightarrow{V^i I_1^i}$ or $\overrightarrow{v_2^i} = \overrightarrow{V^i I_2^i}$. This is illustrated in figure 7.

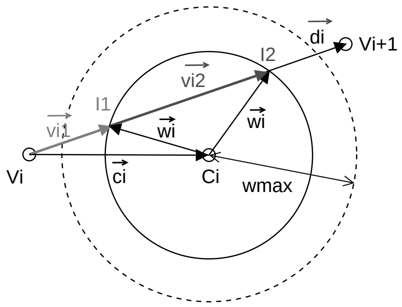


Figure 7: Different possibilities for $\overrightarrow{v^i}$, for $w^i < w_{max}$.

[1] Note that we are sure that at least one intersection exists, because the path $\mathcal{P}$ is supposed to be entirely feasible.

The radius $w^i = w_{max}$ allows to compute the minimal and maximal modulus for $\overrightarrow{v^i}$, denoted v^i_{min} and v^i_{max}:

$$\begin{array}{r}v^i_{min} = \min(v^i_1, v^i_2) \\ v^i_{max} = \max(v^i_1, v^i_2)\end{array} \tag{3}$$

If $\overrightarrow{v^i_j}$ and $\overrightarrow{d^i}$ are not in the same direction, the robot is not moving toward the next viapoint V^{i+1}, but at the opposite (backward move). In this case, to force a forward move, the modulus v^i_j is replaced by 0 in equation 3.

These results allow us to describe the robot's cinematic in presence of currents:

$$\forall i \in [1, n'] : \quad t^i \in [0, T] \qquad (D_{ti})$$

$$v^i \in [v^i_{min}, v^i_{max}] \qquad (D_{vi})$$

$$t^i = t^{i-1} + d^i / v^i \qquad (C_{ti,vi})$$

Note that the quantities d^i, v^i_{min} and v^i_{max} are known and constant:

- The distance d^i is deduced from position of viapoints V^i and V^{i+1},
- The velocity bounds v^i_{min} and v^i_{max} are computed using equation 3.

Therefore, the only variables in the above equations are t^i and v^i (both scalar).

b. Constraints related to moving obstacles avoidance

As explained in section I.2, moving obstacles can be represented in a 2-D space-time (l, t), which l represents the curvilinear abscissa on the path $\mathcal{P}$, and t the elapsed time since departure.

In this space-time, each moving obstacle O^j generates a set of *forbidden surfaces* S^j, containing all forbidden couples (l, t), leading to a collision between the robot and O^j.

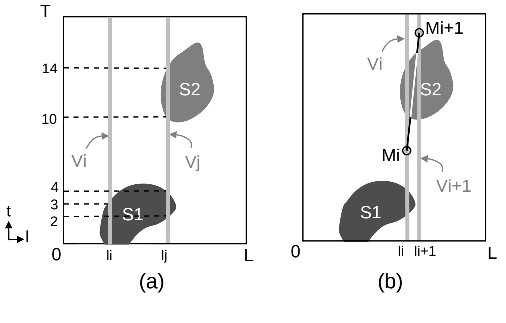


Figure 8: (a) forbidden times for two viapoints V^i and V^j: $F^i = [0, 3]$ and $F^j = [2, 4] \cup [10, 14]$; (b) impossible move between two successive viapoints M^i and M^{i+1}: $M^i \notin F^i$ and $M^{i+1} \notin F^{i+1}$, but $[M^i, M^{i+1}]$ intersects the forbidden surface S^2.

Therefore, for each viapoint V^i, we can define the interval of *forbidden times*, denoted F^i. This interval has the following meaning: if $t^i \in F^i$, then the robot collides a moving obstacle at viapoint V^i. F^i is computed by intersecting all surfaces S^j with the line $l = l^i$.

As shown in figure 8a, F^i is an union of subintervals $F^i_1 \cup F^i_2 \cup ... \cup F^i_{s^i}$, where s^i denotes the number of intersected surfaces.

A first idea to model obstacle avoidance would consist in using the simple constraint:

$$\forall i : \ t^i \notin F^i \tag{4}$$

However, this constraint is too weak to avoid collisions in all cases.

To illustrate this point, let us consider two successive viapoints V^i and V^{i+1}. In the space-time, the visit of these viapoints is symbolized by two points M^i and M^{i+1}. Even if both points respect equation 4, it is not necessary the case for all intermediate points lying on the line segment $[M^i, M^{i+1}]$. Indeed, this line segment can intersect some forbidden surfaces, as shown in figure 8b.

This problem appears when a forbidden surface is by-passed by one side at point M^i and by the other side at point M^{i+1}. In the example of figure 8b, M^i is above the surface S_2, whereas M^{i+1} is below, which leads to an intersection.

A simple way to avoid this situation is to force all the points of the space-time to be on the same side of each forbidden surfaces. This is modeled by the following constraints:

$$\forall i \in [1, n'] : \quad F^i = [\underline{t_1}, \overline{t_1}] \cup ... \cup [\underline{t_{s^i}}, \overline{t_{s^i}}]$$

$$\begin{array}{lll}\forall j \in [1, s^i] : & b_j \in \{0, 1\} & (D_{bj}) \\ & t^i \geq \overline{t_j} - T \cdot (1 - b_j) & (C^1_{ti,bj}) \\ & t^i \leq \underline{t_j} + T \cdot b_j & (C^2_{ti,bj})\end{array}$$

The binary variables b_j allow to represent how the forbidden surface S_j is by-passed. Indeed, $b_j = 1$ if the point M^i is above S_j, else $b_j = 0$.

Since these variables b_j are shared by all points M^i, they are forced to by-pass forbidden surfaces in the same way. Combining the variables b_j and T allows to avoid the use of reification techniques: if one constraint is true, the other is naturally disabled (since $\forall i : \ t^i \leq T$).

2. CSP solving

a. CSP formulation

A CSP is commonly described as a triplet (X, D, C), where:

- $X = \cup\{x^i\}$ is a set of variables,
- $D = \Pi \ D^i$ is a set of domains associated to X (D^i represents the domain of x^i),
- $C = \cup\{C^i\}$ is a set of constraints on elements of X.

Using these notations, our velocity tuning problem can be modeled by the following CSP:

- $X = \cup\{v^i, t^i, b_j\},\ i \in [1, n'],\ j \in [1, s^i]$ where n' is the number of viapoints (including artificial ones) and s^i the number of intersected forbidden surfaces by the line $l = l^i$ in the space-time,
- $D = D_{ti} \times D_{vi} \times D_{bj}$,
- $C = C_{ti,vi} \cup C^1_{ti,bj} \cup C^2_{ti,bj}$.

This CSP has the following properties:

- It contains $2n' + \max_i\{s^i\}$ variables and $n' + 2\max_i\{s^i\}$ constraints. In our applications, $n' < 50$ and $\max_i\{s^i\} < 10$.
- All constraints are linear[2].
- Variables are defined on finite domains, with the following sizes:
  - $|D_{ti}| \approx 1000$ (number of time steps)
  - $|D_{vi}| \approx 100$ (number of different velocities)
  - $|D_{bj}| = 2$ (binary variables)

b. Enumeration strategy

Since many solutions are temporally equivalent, we chose the following enumeration strategy:

- Variables ordering: b_j, then t^i, then v^i (in the decreasing order of i).
- Values ordering:
  - increasing values for b_j (to by-pass the forbidden surface by the bottom first)
  - increasing values for t^i (to determine the first valid time steps)
  - decreasing values for v^i (because $v^i \sim O(1/t^i)$)

With this strategy, we try to visit the viapoints as earlier as possible, from the last viapoint to the first viapoint.

The variables b_j allow to roughly identify a first solution, by determining by which side the forbidden surfaces are bypassed. Then, the variables t^i and v^i refine this solution. Note that the enumeration mainly concern the variables t^i, because a value of t^i imposes a value for v^i.

IV. Extension to other constraints

Modeling the velocity tuning problem as a CSP allows to easily integrate other constraints. This section gives two examples: (1) time-varying currents and (2) temporal constraints.

1. Time-varying currents

In a forecast context, values of currents are valid during a time interval ΔT, depending on the application. For instance, in maritime applications, ΔT represents a few hours.

As for ECAs, we find that it is useless to interpolate these data between two intervals. We thus consider that a time-varying current is defined by successive levels, as shown in figure 9.

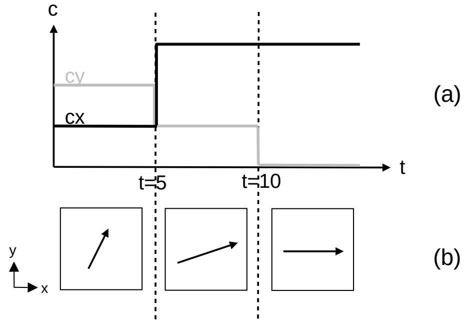


Figure 9: A time-varying current. (a) graph of c_x and c_y functions, defined by levels; (b) the corresponding velocity vector.

Let us consider a current $\overrightarrow{c^i}$, between viapoints V^i and V^{i+1}, changing k times in the interval $[0, T]$. This interval is thus split into $k+1$ subintervals: $[0, t_1]$, $[t_1, t_2]$, ..., $[t_{k-1}, t_k]$, $[t_k, T]$. In each subinterval $[t_j, t_{j+1}]$, the value of the current is constant, denoted $\overrightarrow{c^i_j}$.

The influence of this time-varying current can be modeled in our CSP by using some binary variables. Indeed, the equation (D_{vi}) is replaced by the following constraints:

$$\forall j \in [1, k-1]: \quad \begin{array}{l} b_j \in \{0, 1\} \\ t^i \geq t_j \cdot b_j \\ t^i < (1 - b_j) \cdot T + t_{j+1} \end{array} \tag{5}$$

$$\textstyle\sum_{j=1}^{k-1} b_j = 1 \tag{6}$$

$$v^i \geq \sum_{j=1}^{k-1} b_j \cdot v^i_{min,j} \tag{7}$$

$$v^i \leq \sum_{j=1}^{k-1} b_j \cdot v^i_{max,j} \tag{8}$$

The binary variables b_j allow to identify the subinterval $[t_j, t_{j+1}]$ in which lies the variable t^i. In other terms, $b_j = 1$ if and only if $t^i \in [t_j, t_{j+1}]$. This is modeled by equations 5 and 6.

Then, equations 7 and 8 allows to impose velocity bounds on v^i according to this subinterval. That is, if $t^i \in [t_j, t_{j+1}]$, then $v^i \in [v^i_{min,j}, v^i_{max,j}]$. The values of $v^i_{min,j}$ and $v^i_{max,j}$ are computed as explained in part III.1a, substituting $\overrightarrow{c^i}$ by $\overrightarrow{c^i_j}$.

This model is simple but rough. More precisely, it ignores current changes between two successive viapoints. Therefore, an error is potentially made on velocity bounds. This error remain negligible if the distance d^i between viapoints is small.

[2] After the change of variable $v^{i'} = 1/v^i$.

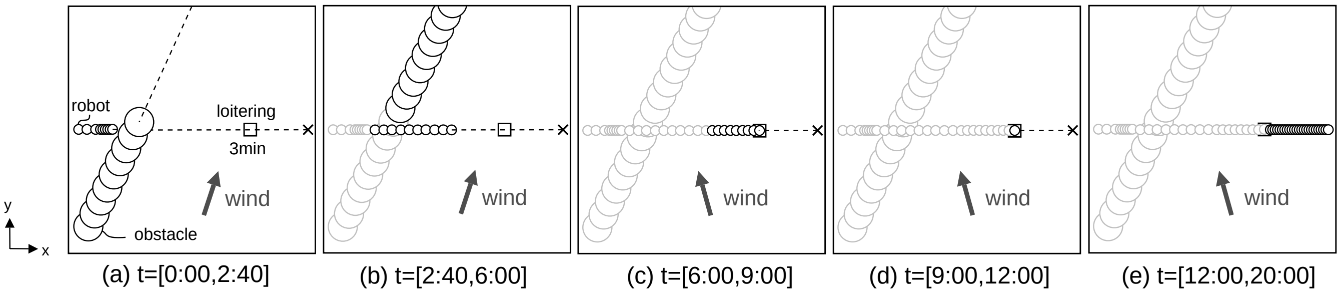


Figure 10: Complete example: (a)(b) moving obstacle avoidance, (c) effect of a current change at $t = 6\text{min}$, (d) loitering during $D = 3\text{min}$ (in black square) and (e) effect of a time window, imposing the arrival at $t = 20\text{min}$.

If it is not the case, d^i can be reduced by artificially subdividing ECAs. By this way, the size of ECAs is decreased and the number of artificial viapoints increased. Therefore, viapoints will be globally closer from each other.

2. Temporal constraints

In this section, we explain how to temporally constrain a viapoint V^i. Especially, we study two temporal constraints particularly mentioned in literature: (a) time windows and (b) loitering.

a. Time windows

A time window W^i is a couple $(\underline{w^i}, \overline{w^i})$, specifying the minimum date $\underline{w^i}$ and the maximum date $\overline{w^i}$ for the robot to visit the viapoint V^i.

In a military context, by example, time windows may correspond to strategic data, such as: "the target will be at V^i between $\underline{w^i}$ and $\overline{w^i}$".

Modeling of W^i is quite natural in our CSP, leading to the single constraint:

$$t^i \in [\underline{w^i}, \overline{w^i}]$$

b. Loitering

The concept of loitering consists in forcing the robot to wait at viapoint V^i for a given duration D^i. From a practical point of view, D^i may correspond to the minimum time required to perform a task at V^i.

Here, our goal does not consist in choosing the best value of D^i, but choosing the best beginning time t^i for the loitering task.

This choice seems to be hard, because it depends both on the moving obstacles and the current changes. However, it can be simply modeled in our CSP, replacing the constraint $(C_{ti,vi})$ by :

$$t^i = t^{i-1} + d^i/v^i + D^i \tag{9}$$

V. Experimental results

This section has two objectives: (1) illustrating our approach and (2) evaluating its performance.

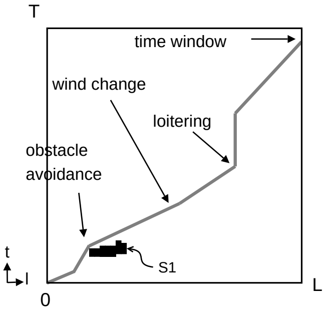


Figure 11: The space-time corresponding to fig. 10

1. Illustrative example

We illustrate here all the constraints presented before through a complete example containing: a moving obstacle, a current change, a loitering task and a time window on arrival.

In this example, simple instances of the constraints have been chosen: (1) the current is uniform on the map and (2) the moving obstacle performs a straight-line move at constant velocity.

The result obtained by our approach is depicted in figures 10 and 11. Figure 10 shows the different phases of velocity tuning in the initial environment, and figure 11 in the space-time.

2. Performance evaluation

In this part, we evaluate experimentally the impact of current changes and moving obstacles on the computation time, in the following conditions:

- **Hardware**: Our approach has been run on a $1.7Ghz$ PC with $512Mo$ of RAM, using the `clpfd` library (Carlsson, Ottosson, & Carlson 1997), provided by Sicstus.
- **Current data**: All data are issued from real wind charts, collected daily during three months on Meteo France website[3] (leading to about 90 different charts). The wind changes are simulated as follows: to simulate k wind

[3] http://www.meteofrance.com/FR/mer/carteVents.jsp

changes, the interval $[0, T]$ is divided into $k + 1$ equal subintervals. A different wind chart is used for each subinterval.

- **Moving obstacles**: As in figure 10, each moving obstacle goes across the environment by performing a straight-line move $P_1 \rightarrow P_2$ at constant velocity. This move is computed in the following way:

  1. Two points P_1 and P_2 are randomly chosen on two borders of the environment, until an intersection I between the path $\mathcal{P}$ and the line segment $[P_1, P_2]$ is detected.
  2. The velocity of the obstacle is chosen such that the obstacle and the robot are at the same time at point I.

The resulting computation times are provided in table 1. Each cell is the mean time obtained on 100 different environments.

Table 1: Average computation time (in ms), for m moving obstacles and k current changes .

m k	0	1	2	3	4	5	6
0	5	9	11	14	17	21	26
1	7	12	13	16	20	24	27
2	10	14	15	18	23	28	29
3	16	21	23	25	34	35	38
4	51	55	56	68	66	67	71
5	80	97	104	106	111	112	114
6	98	127	147	152	159	162	166

From a strictly qualitative point of view, we can observe that the global computation time remains reasonable (a few milliseconds) even in complex environments. Therefore, we think that our approach is potentially usable in on-boards planners.

A theoretical study of the time complexity could confirm these results. In particular, it could be interesting to try different enumeration strategies and evaluate their impact on computational performances.

Conclusion

In this paper, we proposed a velocity tuning approach, based on Constraint Logic Programming (CLP). At our knowledge, this approach is the first able to handle currents. Moreover, this approach is computationally efficient and flexible.

Indeed, we explained that modeling the velocity tuning problem into a Constraint Satisfaction Problem (CSP) allows to easily incorporate more complex constraints, in particular time-varying currents. Moreover, our experiments showed the velocity tuning task could be performed in a polynomial time. It means that our approach is potentially usable in on-board planners.

Further works will investigate the coordination of multiple robots sharing the same environment. In particular, we will study how additional constraints could allow the coordination of fleets of UAVs (Unmanned Air Vehicles).

Acknowledgments

The authors would like to thank Paul-Edouard Marson, Maxime Chivet, Nicolas Vidal and Katia Potiron for their careful reading of this paper.

References

Borrow, J. E. 1988. Optimal robot path planning using the minimum-time criterion. *Journal of Robotics and Automation* 4:443–450.

Canny, J. 1988. *The Complexity of Robot Motion Planning*. MIT Press.

Carlsson, M.; Ottosson, G.; and Carlson, B. 1997. An open-ended finite domain constraint solver. In *Proceedings of Programming Languages: Implementations, Logics, and Programs*.

Fortune, S. 1986. A sweepline algorithm for voronoi diagrams. In *Proceedings of the second annual symposium on Computational geometry*, 313–322.

Garau, B.; Alvarez, A.; and Oliver, G. 2005. Path planning of autonomous underwater vehicles in current fields with complex spatial variability: an a^* approach. In *Proceedings of the International Conference on Robotics and Automation*, 194–198.

Ju, M.-Y.; Liu, J.-H.; and Hwang, K.-S. 2002. Real-time velocity alteration strategy for collision-free trajectory planning of two articulated robots. *Journal of Intelligent and Robotic Systems* 33:167–186.

Kant, K., and Zucker, S. W. 1986. Toward efficient trajectory planning: the path-velocity decomposition. *The International Journal of Robotics Research* 5:72–89.

Khatib, O. 1986. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the International Conference on Robotics and Automation*, volume 2, 500–5005.

LaValle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. *TR 98-11, Computer Science Dept., Iowa State Univ*.

Nilsson, N. J. 1969. A mobile automation: An application of artificial intelligence techniques. *Proceedings of the International Joint Conference on Artifical Intelligence* 509–520.

Park, S.; Deyst, J.; and How, J. 2004. A new nonlinear guidance logic for trajectory tracking. *Proceedings of the AIAA Guidance, Navigation and Control Conference*.

Petres, C.; Pailhas, Y.; Patron, P.; Petillot, Y.; Evans, J.; and Lane, D. 2007. Path planning for autonomous underwater vehicles. *Transactions on Robotics* 23:331–341.

Soulignac, M., and Taillibert, P. 2006. Fast trajectory planning for multiple site surveillance through moving obstacles and wind. In *Proceedings of the Workshop of the UK Planning and Scheduling Special Interest Group*, 25–33.

Zhao, Q., and Yan, S. 2005. Collision-free path planning for mobile robots using chaotic particle swarm optimization. In *Proceedings of the International Conference on Advances in Natural Computation*, 632–635.

SHORT PAPERS

Planning as a software component: A Report from the trenches [*]

Olivier Bartheye and Éric Jacopin
MACCLIA
Crec Saint-Cyr
Écoles de Coëtquidan
F-56381 GUER Cedex
{olivier.bartheye,eric.jacopin}@st-cyr.terre.defense.gouv.fr

An awarded claim While the Pengi paper (AGRE & CHAPMAN 1987) received a Classic Paper award at AAAI'2006 (News 2006), to our knowledge we have yet to see whether its main claim on classical planning is true (AGRE & CHAPMAN 1987, page 269): that "*a traditional problem solver for the Pengo domain [could not cope] with the hundreds or thousands of such representations as* (AT BLOCKS-213 427 991), (IS-A BLOCK-213 BLOCK), *and* (NEXT-TO BLOCK-213 BEE-23)". Or, stated differently (AGRE & CHAPMAN 1987, page 272): "*[The Pengo domain] is one in which events move so quickly that little or no planning is possible, and yet in which human experts can do very well.*"

The Pengo domain is that of a video-game of the eighties where a player navigates a penguin around a two dimensional maze of pushable ice blocks. The player must collect diamonds distributed across the maze while avoiding to get killed by bees; but the player can push an ice block which kills a bee if it slides into it.

The Pengi system described in the Pengi paper (AGRE & CHAPMAN 1987) is a video-game playing system which just happens to fight bees in the Pengo game. Pengi first searches for the penguin on the screen to register its initial position. Then searches for the most dangerous bee, an appropriate weapon to kill that bee (that is, an ice block) and then navigate the penguin towards that weapon to kick it. Both written in Lisp, the Pengo game and the Pengi system are in fact the same Lisp program: the search for the penguin and the most dangerous bee can be made directly by looking at the Lisp data structures. According to the on-going conditions of the game, various pieces of code are activated (for instance, you may wish to push an ice block several times before it becomes a weapon). We refer the reader to the Pengi paper for further information on the Pengi system. Finally, "*[Pengi] plays Pengo badly, in near real time. It can maneuver behind blocks to use as projectiles and kick them at bees and can run from bees which are chasing it*" (AGRE & CHAPMAN 1987, page 272).

Interpreted as a finite state machine, the Pengi system can easily be re-implemented and not only fight bees not badly but also collect diamonds even in non trivial mazes (DROGOUL, FERBER, & JACOPIN 1991).

The awarded claim eventually is about space and time complexity in the Pengo domain and of classical planning algorithms around 1987. But since 1987, processors are several hundred times faster and fastest classical planners are able to produce plans with hundreds of actions in a matter of seconds for certain problems. Consequently, we thought it would be interesting and, most surely, fun, to see how the current technology could cope with an 1980s video-game.

We here report on our very first steps towards the evaluation of the claim about classical planning.

Classical planning, really? As a testbed, we chose Iceblox (BARTLETT, SIMKIN, & STRANC 1996, pages 264–268), a slightly different version of the Pengo game for which there exists an open and widely available java implementation (HORNELL 1996). For instance (cosmetic differences): flames, and not bees, are chasing the penguin-player who must now collect coins, and not diamonds. Moreover (different actions), coins must be extracted from ice blocks. Extraction means kicking seven time at an ice block to destroy the ice and thus making the coin ready for collection. Such an ice block with a coin inside slides as well as any other ice block. So the player must kick in a direction where the ice block cannot slide (e.g. against the edge of the game) in order to extract an iced coin.

Instead of designing a new planning system, we decided to pick up an existing one, and eventually several, in order to compare their relative performance if they had any ability at playing Iceblox. We consequently decided to re-implement Iceblox in Flash (Adobe 2007). Not only would we provide a new implementation of the game, but also could we use the plug-in architecture of the Flash runtime: a call and return mechanism can run (and pass in and out parameters to) any external piece of executable code when put in the appropriate directory.

This deviates from the original Pengi system which was the same Lisp program as the Pengo game (and also deviates from (DROGOUL, FERBER, & JACOPIN 1991) where everything was implemented in the same SmallTalk program), but would eventually ease the comparison as classical planners are not necessarily written in Flash.

However, this dramatically changes the setting of the problem.

[*] Special thanks to Maria FOX, Jörg HOFFMANN, Jana KOEHLER and Derek LONG about the gripper domain.

On one side, a classical planner becomes an external component which happens to provide a planning functionality: fine, that's how we want it to work.

On the other side, the world view of the Pengi paper (AGRE 1993) is that of the dynamics of everyday life (AGRE 1988) (plans do exist, but are better communicated[1] to people than built from scratch) and thus is opposed to the heavily intentional (BRATMAN 1987; MILLER, GALANTER, & PRIBRAM 1986) world view of planning.

In other words: the Pengi system is always in charge of the actions (moving the penguin, kicking ice blocks) whereas an external component is in charge only when activated and is harmless otherwise: a player must be able to play Iceblox when the planning component is not activated or no component is plugged-in. This generates supplementary questions: when is classical planning activated and for how long? One more constraint. To respect the dynamics of the domain of video-games, Iceblox must never stop and must run while the classical planning component is planning: flames keep on chasing the penguin and sliding ice blocks keep on sliding.

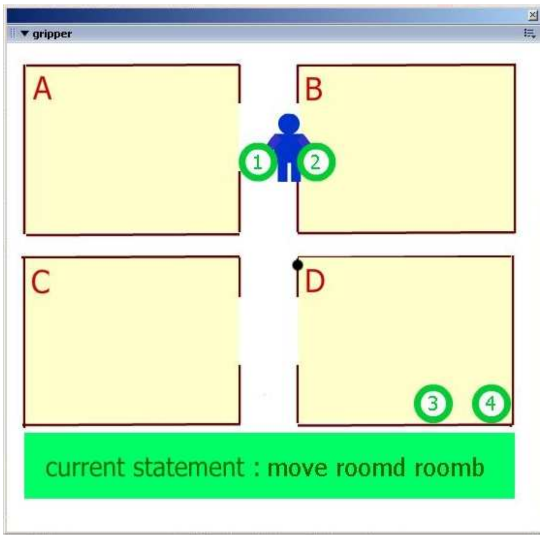
Consequently, the classical planning component is activated when the player presses the "*p*" key. This activation is ended as soon as the player presses the keyboard again: the arrow keys to move the penguin up, right, down and left; and the space key to kick an ice block.

Hopefully, an anonymous classical planner shall build a plan and return it to Iceblox. What shall Iceblox do with this plan? Please, note that this question does not immediately entail further questions of interleaving classical planning and execution (AMBROS-INGERSON & STEEL 1988). To begin with, there is a matter of level of detail: actions in Iceblox corresponds to keys pressed by the player. Is the classical planning component really expected to build plans with such actions?

Hints from a gripper video game On one hand, the classical planning component is expected to build plans with keys pressed. First because it seems part of the claim: if the classical planning component (that is, the "*traditional problem solver*" of the claim) has to cope "with hundreds or thousands" of detailed representations describing the initial and final situations, then we can expect action representations to be as detailed as the initial and final situations. However, the Pengi literature (AGRE & CHAPMAN 1987; AGRE 1988; 1993; 1997; CHAPMAN 1990) says nothing about this.

On the other hand, classical planners are used to cope with high-level action description. For instance, here is the classical planning Move operator from the well-known gripper (FOX & LONG 1999) domain:

$$\text{Move(X,Y)} = \left\{ \begin{array}{lcl} \textbf{Preconditions} & : & \{\text{at-robby(X)}\} \\ \textbf{Additions} & : & \{\text{at-robby(Y)}\} \\ \textbf{Deletions} & : & \{\text{at-robby(X)}\} \end{array} \right.$$



current statement : move roomd roomb

Figure 1: An anonymous classical planning system has built (actually, it's FF, plugged-in our Flash application as described earlier; but let's say we didn't tell you) a plan for the following the gripper video game problem: 4 balls must be moved from room B to room D. The on-going action (from the plan) is printed in the green area at the bottom of the window: robby is moving from room D to room B; details of the navigation (and of the picking up and down of balls) are left to the Flash application.

In the gripper domain robby-the-robot uses its arms to move balls from one room, along a corridor, to another. Neither bees nor flames prevent robby-the-robot from succeeding in transporting balls from one room to another. It is nevertheless easy to come up with a simplistic two dimensional gripper video-game: your task is to move as fast as possible a set of balls from their initial location to their final location (see Figure 1).

As stupid as this may sound, this gripper video-game isn't too far from, say, the popular Sokoban video-game (in a maze, blocks must be slided from one place to another, with no time limit) (CHARRIER 2007). In such a puzzle, the details of the block pushing activity are important: e.g. a wrong push at a corner can make the problem unsolvable. But more important is the block you push next, which sequences the player's next Move. Similar Iceblox situations where the player only needs to navigate towards iced coins and then extract them do exist (See Figure 2).

Here are two operators which can combine into a plan and solve the simple situation of Figure 2: first MoveToCoin(6,4), then Extract(6,4).

[1] Official player's guides are good sources of plans communicated to video-game players that would otherwise take some time to build.

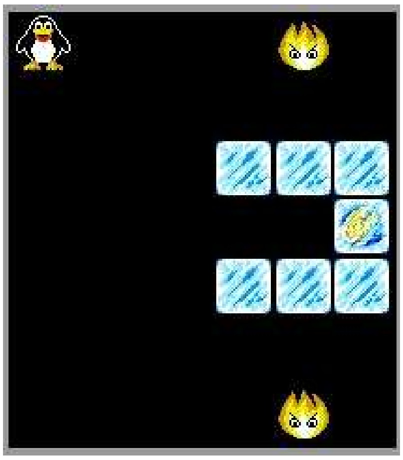


Figure 2: A simplistic level in the Iceblox domain: move to the ice block containing a coin and then extract it. Details of the navigation, as far as possible from the flames, and of the extraction of the coin (seven kicks to the ice block) are again left to the Flash application.

$$\text{MoveToCoin(X,Y)} = \begin{cases} \textbf{Preconditions :} & \{\text{at(X,Z)}\} \\ \textbf{Additions :} & \{\text{at-coin(X,Y)}\} \\ \textbf{Deletions :} & \{\text{at(X,Z)}\} \end{cases}$$

$$\text{Extract(X,Y)} = \begin{cases} \textbf{Preconditions :} & \{\text{at-coin(X,Y),} \\ & \quad \text{iced-coin(X,Y)}\} \\ \textbf{Additions :} & \{\text{at(X,Y),} \\ & \quad \text{extracted(X,Y)}\} \\ \textbf{Deletions :} & \{\text{at-coin(X,Y),} \\ & \quad \text{iced-coin(X,Y)}\} \end{cases}$$

Since we have neither implemented flame-fighting nor fleeing operators, flames must be un-aggressive so that the coin of Figure 2 can be extracted. And because of the simple path from the Penguin to the coin, the initial and final situations are simply described: {at(1,1), iced-coin(6,4)} and {extracted(6,4)}, respectively.

We won't discuss this extremely low number of formulas needed to describe what could be called a *minimal* Iceblox problem: up to now, the biggest part of our work has been devoted to stay as close as possible to the spirit of classical planning and video-games, while designing a satisfying testbed. In the future, we hope to concentrate more on designing classical planning predicates and operators in order to cope with more complex Iceblox situations.

References

Adobe. 2007. Flash. `http://www.adobe.com/`.

News, A. 2006. Classic paper award. *AI Magazine* ***27****(3)* 4.

Agre, P., and Chapman, D. 1987. Pengi: An implementation of a theory of activity. In *Proceedings of* 6^{th} *AAAI*, 268–272.

Agre, P. 1988. *The Dynamics of Everyday life*. Ph.D. Dissertation, MIT AI Lab Tech Report 1085.

Agre, P. 1993. The symbolic worldview: Reply to vera and simon. *Cognitive Science* ***17****(1)* 61–69.

Agre, P. 1997. *Computation and Human Experience*. Cambridge University Press.

Ambros-Ingerson, J., and Steel, S. 1988. Integrating planning, execution and monitoring. In *Proceedings of* 7^{th} *AAAI*, 83–88.

Bartlett, N.; Simkin, S.; and Stranc, C. 1996. *Java Game Programming*. Coriolis Group Books.

Bratman, M. 1987. *Intentions, Plans and Practical Reason*. Harvard University Press.

Chapman, D. 1990. *Vision, Instruction and Action*. Ph.D. Dissertation, MIT AI Lab Tech Report 1204.

Charrier, D. 2007. Super sokoban 2.0. `http://d.ch.free.fr/`.

Drogoul, A.; Ferber, J.; and Jacopin, E. 1991. Viewing cognitive modelling as eco-problem solving: The pengi experience. In *Proceedings of the 1991 European Conference on Modelling and Simulation Multiconference*, 337–342.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *Proceedings of* 16^{th} *IJCAI*, 956–961.

Hornell, K. 1996. Iceblox. `http://www.tdb.uu.se/~karl`.

Miller, G.; Galanter, E.; and Pribram, K. 1986. *Plans and the Structure of Behavior*. Adams-Bannister-Cox.

Nurse Scheduling Web Application

Zdeněk Bäumelt^{1,2}**, Přemysl Šůcha**^{1}**, Zdeněk Hanzálek**^{1,2}

^{1}Department of Control Engineering, Faculty of Electrical Engineering
Czech Technical University in Prague, Czech Republic, {baumez1, suchap, hanzalek}@fel.cvut.cz

^{2}Merica s. r. o., Czech Republic, {imedica, hanzalek}@merica.cz

Abstract

The focus of this paper is on the development of a web application for solving Nurse Scheduling Problem. This problem belongs to scheduling problems domain, exactly timetabling problems domain. It is necessary to consider large amount of constraints and interactions among nurses, that can be simplified through web access.

Introduction

Preparation of multishift schedule is rather difficult process which incorporates couple of constraints (e.g. minimum number of nurses for each type of shift, nurses' workload, balanced shift assignment) and interaction of several users (nurses' requests consideration). Even though single-user nurse scheduling applications avoid rather painful manual process, they do not allow easy access of all nurses to interact with each other. This problem can be efficiently solved using modern web technologies, while carefully considering all specific features of such application; e.g. large amount of human interactions, dramatic impact on satisfaction of individual nurse as well as good mood in nurse team.

Definition of Nurse Scheduling Problem

Nurse Scheduling Problem (NSP) is NP-hard problem, that belongs to timetabling or personnel scheduling domain. The solution of this problem should satisfy all constraints, that are set on the input. With larger instances (growing with number of nurses, number of days in schedule, set of constraints) NSP comes to the combinatorial explosion and it is harder to find an optimal solution.

Related Works

There are several views for solving NSP. In background paper (Hung 1995) there is a history of NSP research from the 60's to 1994. Other bibliographic survey with one described approach is in (Cheang *et al.* 2002). More actual survey is presented in (Burke *et al.* 2004).

On one hand, there is the branch of optimal solution approaches. It includes linear programming (LP) and integer linear programming (ILP) (Eiselt & Sandblom 2000). On the other hand, there are some heuristic approaches. One way to find some solution is to use artificial intelligence methods (e.g. declarative and constraint programming (Okada 1992) or expert systems (Chen & Yeung 1993)). The second way is to use some metaheuristics (simulated annealing, tabu search (Berghe 2002) or evolutionary algorithms (Aickelin 1999)).

Contributions

This paper uses Tabu Search approach and the main contribution of this work lies in application structure designed for access via web.

Application Structure

The structure of Nurse Scheduling Web Application (NSWA) is shown in Figure 1. Users can work with the application via common web browsers. All application blocks are on the server side, which brings many other advantages (operating system independence, no installation and upgrades on client side). The scheduling algorithm runs independently as a web service and exchanges data with application and database through communication interface.

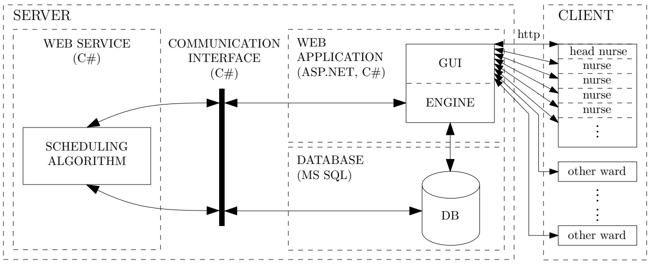


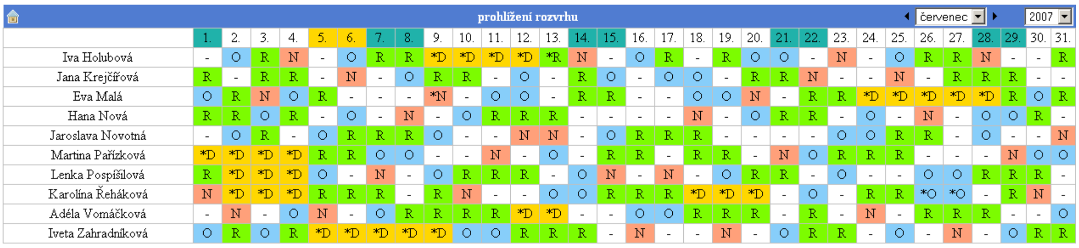
Figure 1: NSWA structure - block design.

Scheduling Algorithm

We decided to use a scheduling algorithm that is based on multicriterial programming implemented as Tabu Search metaheuristic.

Mathematical Model

Our mathematical model is designed as three-shift model – early (E), late (L) and night shift (N) (in Figure 2 early (R),



prohlížení rozvrhu červenec 2007

	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.	16.	17.	18.	19.	20.	21.	22.	23.	24.	25.	26.	27.	28.	29.	30.	31.
Iva Holubová	-	O	R	N	-	O	R	R	*D	*D	*D	*D	*R	N	-	O	R	-	R	O	O	-	N	-	O	R	R	N	-	-	R
Jana Krejčířová	R	-	R	R	-	N	-	O	R	R	-	O	-	R	O	-	O	O	-	R	R	N	-	-	N	-	R	R	R	-	-
Eva Malá	O	R	N	O	R	-	-	-	*N	-	O	O	-	R	R	-	-	O	O	N	-	R	R	*D	*D	*D	*D	*D	R	O	R
Hana Nová	R	R	O	R	-	O	-	N	-	O	R	R	R	-	-	-	-	N	-	O	R	R	-	O	-	N	-	O	O	R	-
Jaroslava Novotná	-	O	R	-	O	R	R	R	O	-	-	N	N	-	O	R	R	R	-	-	-	-	O	O	R	R	-	O	-	-	N
Martina Pařízková	*D	*D	*D	*D	R	R	O	O	-	-	N	-	O	-	R	R	-	R	R	-	N	O	R	R	R	-	-	-	N	O	O
Lenka Pospíšilová	R	*D	*D	*D	O	-	N	-	O	R	R	R	-	O	N	-	N	-	O	R	R	-	O	-	-	O	O	R	R	R	-
Karolína Řeháková	N	*D	*D	*D	R	R	R	-	R	N	-	-	O	O	R	R	R	*D	*D	*D	-	O	-	R	R	*O	*O	-	R	N	-
Adéla Vomáčková	-	N	-	O	N	-	O	R	R	R	R	*D	*D	-	-	O	O	R	R	R	-	R	-	N	-	R	R	R	-	-	O
Iveta Zahradníková	O	R	O	R	*D	*D	*D	*D	*D	O	O	R	R	R	-	N	-	-	N	-	O	R	R	-	O	-	N	-	O	R	R

Figure 2: Screenshot from our Nurse Scheduling Web Application (july 2007).

late (O), night shift (N), holiday (D) – shifts with star are requested shifts by nurses). Coverage is per shift, under and over coverage is not allowed. We decided for one month scheduling period, because of data export to salary administration. There are no qualification groups (all nurses have the same qualification) and we considered full-time workload in this version of mathematical model.

We optimize objective function Z

$$\min_{x \in X}(Z(x)) \tag{1}$$

where x is one schedule from X state space of schedules. There are two types of constraints in our mathematical model.

- hard constraints have to be fulfilled always
- soft constraints with penalization $f_j(x)$ that are the subject of objective function $Z(x)$, which is defined as

$$Z(x) = \mathbf{w} \cdot \mathbf{f}(x) = \sum_{j=1}^{d} w_j f_j(x), \qquad w_j \geq 0, d = \dim(\mathbf{f}(x)) \tag{2}$$

where $\mathbf{w}$ is a vector of weights given by user, $\mathbf{f}(x)$ is a vector function of constraints penalization and d is a number of soft constraints. In our algorithm we considered the following constraints:

Hard Constraints

- required number of nurses for each shift type (#RE, #RL, #RN)
- to consider days from previous month (#H)
- nurses' requests consideration (#R)
- one shift assignment per day (hc1a)
- no early shift after night shift assignment (hc1b)
- no more than five consecutive working days (hc2)
- forbidden shift combinations (FC)

Soft Constraints

- nurses' work-load balance (sc1)
- nurses' day/night shift balance (sc2)
- nurses' weekend work-load balance (sc3)
- avoiding isolated working days (sc4)
- avoiding isolated days-off (sc5)

Head nurses can choose which of hard constraints will be used in our algorithm. Soft constraints are weighted by the head nurses as well. Some hard constraints (hc2, FC) have been converted to the soft constraints with very large weights compared to weights of soft constraints sc1, sc2, sc3, sc4, sc5.

The outline of full Nurse Scheduling Algorithm is described in Algorithm 1 below.

Algorithm 1 – Nurse Scheduling Algorithm

```
1. read the scheduling parameters and the nurses requests;
2. find a feasible solution $x_{init}$ satisfying hard constraints;
3. optimization (Algorithm 2);
4. user choice
   • schedule is acceptable, goto 7;
   • schedule is acceptable with manual corrections, goto 6;
   • schedule is not acceptable, user can reconfigure scheduling
     parameters, goto 5;
5. reconfiguration of scheduling parameters, goto 1, 3 or 6;
6. manual corrections, goto 3, 5 or 7;
7. end of optimization, save the schedule.
```

Tabu Search Algorithm

Tabu Search algorithm shown in detail in Algorithm 2 is used to reduce the state space of schedules.

In our implemenentation, $TabuList$ represents the list of forbidden shift exchanges and has three attributes. The indexes i_1 and i_2 represent origin and target row (nurse) of shift exchange. The third index j is day index. Length of $TabuList$, so called $TabuList\ tenure$, was set to 8.

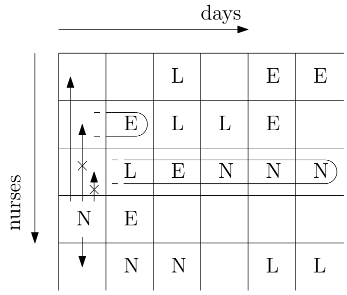


Figure 3: The *candidate* search, non-permissible shift exchanges.

Table 1: NSWA experiments.

n	m	#RE	#RL	#RN	#H	#R	hc1	hc2	FC	w(sc1)	w(sc2)	w(sc3)	w(sc4)	w(sc5)	$t_s[s]$
28	12	4	3	2	0	0	1	1	0	0	0	0	0	0	1.336
28	12	4	3	2	5	0	1	1	0	0	0	0	0	0	2.396
28	12	4	3	2	5	1	1	1	0	0	0	0	0	0	1.038
28	12	4	3	2	5	0	1	1	0	100	100	0	0	0	2.860
28	12	4	3	2	5	0	1	1	0	100	100	100	100	100	4.342
28	12	4	3	2	5	0	1	1	'NNLL'	100	100	100	100	100	6.460
28	12	4	3	2	5	0	1	1	'NNLL' 'LNLE'	100	100	100	100	100	7.327
28	20	7	5	3	5	0	1	1	'NNLL' 'LNLE'	100	100	100	100	100	88.588
28	20	3	2	2	5	0	1	1	'NNLL' 'LNLE'	100	100	100	100	100	35.607

Let the $candidate$ be a possible shift exchange in one day that satisfies hard constraints (see Figure 3 – two candidates are forbidden due to hard constraints hc1b, hc2). Let x_{cand} be the schedule x within updated $candidate$ shift exchange and $Z(x_{cand})$ be the value of objective function of this schedule.

Algorithm 2 – Tabu Search Algorithm

```
1. compute $Z(x_{init})$;
2. $x := x_{init}$;              $x_{next} := x_{init}$;
   $Z(x) := Z(x_{init})$;        $Z(x_{next}) := Z(x_{init})$;
3. while (($Z(x) > 0$) & ($\exists$ not forbidden $f_j(x)$)) (Figure 4)
     choose $\max(w_j f_j(x))$, $j \in$ not forbidden constraints;
     for $\forall candidate$
       if ($candidate \notin TabuList$)
         compute $Z(x_{cand})$;
         if ($Z(x_{cand}) < Z(x_{next})$)
           $x_{next} := x_{cand}$;
           $Z(x_{next}) := Z(x_{cand})$;
         endif
       endif
     endfor
     if ($Z(x_{next}) < Z(x)$)
       $x := x_{next}$;
       $Z(x) := Z(x_{next})$;
       add opposite exchange to the top of $TabuList$;
       clear all forbidden constraints (Figure 4, step 2);
     else
       add an empty record to the top of $TabuList$;
       forbid the chosen constraint (Figure 4, steps 1, 3, 4, 5, 6);
     endif
   endwhile
4. return $x$, $Z(x)$.
```

Let $next$ be the best $candidate$ (with respect to the objective function) at each optimization step. When we have gone through all possible $candidates$, we compare values of $Z(x)$ and $Z(x_{next})$ and choose the better one for the next step of optimization. The idea of soft constraint choice and algorithm termination is demonstrated in Figure 4 for the case of four soft constraints.

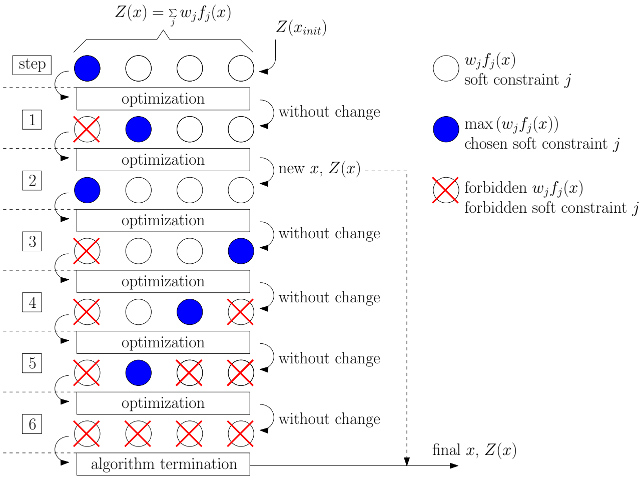


Figure 4: Choice of soft constraints for the next step of optimization and algorithm termination.

Experiments

We used our NSWA called iMEDICA[1] for the instances, that are presented Table 1. Columns from n to w(sc5) are input parameters (n stand for the number of nurses and m for number of days in schedule, other columns are hard and soft constraints). Column FC shows considered forbidden shift combinations (e.g. 'LNLE' - late, night, late and early shift). Output parameter t_s is scheduling time in seconds including steps 1-4 from Algorithm 1 and web communication. The instances were computed on server Intel Pentium 3.4 GHz@4 GB DDR.

In order to evaluate our NSWA, we implemented optimal solution via ILP for simplified two-shift type (day and night) mathematical model (Azaiez & Sharif 2005). We used free solver GLPK[2]. Scheduling times for instances with $n \sim 10$ nurses and $m = 28$ days were hundreds of seconds (more results are in (Baumelt 2007)).

[1] iMEDICA, `http://imedica.merica.cz/`, the product of Merica s. r. o.

[2] GPLK, `http://www.gnu.org/software/glpk/`

Conclusions

In this paper we briefly presented our Nurse Scheduling Web Application. We have got the feedback from several hospitals in the Czech Republic. In cooperation with these hospitals we are working on the improvement of the mathematical model and application interface.

Acknowledgements

This work was supported by the Ministry of Education of the Czech Republic under Project 2C06017.

References

Aickelin, U. 1999. *Genetic Algorithms for Multiple-Choice Optimisation Algorithms*. Ph.d. diss., European Business Management School University of Swansea.

Azaiez, M. N., and Sharif, S. S. 2005. A 0-1 goal programming nodel for nurse scheduling. *Computers & Operations Research* 32.

Baumelt, Z. 2007. *Hospital Nurse Scheduling*. Master thesis, Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague.

Berghe, G. V. 2002. *An Advanced Model and Novel Meta-Heuristic Solution Methods to Personnel Scheduling in Healthcare*. Ph.d. diss., University of Gent.

Burke, E. K.; de Causmaecker, P.; Berghe, G. V.; and van Landeghem, H. 2004. The state of the art of nurse rostering. *Journal of Scheduling* 7:441–499.

Cheang, B.; Li, B.; Lim, A.; and Rodrigues, B. 2002. Nurse rostering problems – a bibliographic survey. *European Journal of Operations Research*.

Chen, J., and Yeung, T. 1993. Hybrid expert system approach to nurse scheduling. *Computers in Nursing*.

Eiselt, H. A., and Sandblom, C. L. 2000. *Integer Programming and Netwotk Models*. Springer-Verlag Berlin and Heidelberg, 1st edition.

Hung, R. 1995. Hospital nurse scheduling. *Journal of Nursing Administration* 25.

Okada, M. 1992. An approach to the generalised nurse scheduling problem – generation of a declarative program to represent institution-speciffic knowledge. *Computers and Biomedical Research* 25.

PSPSolver: An Open Source Library for the RCPSP

Javier Roca, Filip Bossuyt
Intelligent Software Company
Chaussée de Nivelles 121/2, 7181 Arquennes, Belgium

Gaetan Libert
Computer Science Department - Faculté Polytechnique de Mons
Rue de Houdain 9, 7000 Mons, Belgium

{javier.roca, filip.bossuyt}@planningforce.com, gaetan.libert@fpms.ac.be

Abstract

The Resource-Constrained Project Scheduling Problem (RCPSP) is a classical problem in project scheduling. The most common and successful approaches to solve the RCPSP are those applying heuristics, metaheuristics and sampling schemas, given their practicability and effectiveness. In most of the cases these approaches apply a Schedule Generation Schema (SGS) combined with a suitable solution representation and priority rules. Although there is a considerable research in these RCPSP solving methods and its theory there is a lack of software for supporting the research of new solving methods. In many cases, the RCPSP research requires the implementation of algorithms in order to validate or evaluate a solving method. We introduce PSPSolver (Project Scheduling Problem Solver), an extensible and practical heuristic-based library for supporting the research on solvers for the RCPSP.

1.The RCPSP Model

Informally, the single mode RCPSP model, simply referred as RCPSP, is a well known project scheduling problem (PSP) that seeks the answer to the following question: *"Given the limited availability of resources, what is the best way to schedule the activities in order to complete the project in the shortest possible time?"*. The RCPSP is of special interest in fields like construction and production scheduling. Conceptually, the RCPSP is a PSP with single mode activities, renewable constrained resources, finish-to-start precedence relationships with zero time lags, no pre-emption, and has the makespan minimization as the performance measure. According to Bucker et al. [1], this problem is denoted as *PS | prec | Cmax* (machine scheduling domain). Herroelen et al. [2] denotes this model as *m,1 | cpm | Cmax*.

Due to the fact that the RCPSP forms the core problem among the class of resource-constrained project scheduling problems [1], every improvement in its resolution can produce new advances in the resolution of the other models. The RCPSP instances are usually represented as A-O-N digraphs (Figure 1), while the RCPSP solutions (schedules) are represented as Gantt charts (Figure 2).

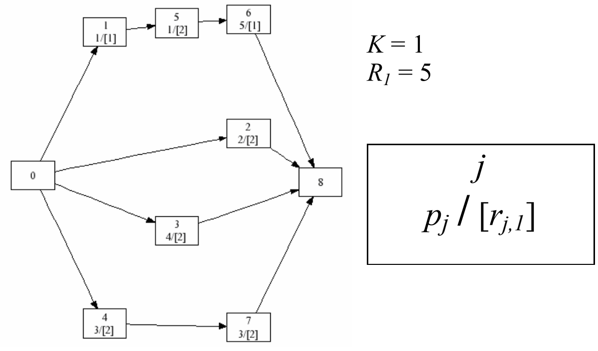


Figure 1: A RCPSP Instance

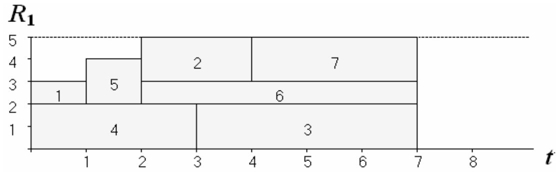


Figure 2: An optimal schedule for the instance in Figure 1.

2. Solving the RCPSP

The main approaches to solve the RCPSP are the optimal (exact) methods, heuristics, and the metaheuristics-based solution procedures. It has been shown by Blazewicz et al. [3] that the RCPSP, as a generalization of the classical job shop scheduling problem, belongs to the class of *NP*-hard optimization problems. Therefore, heuristic solution procedures are indispensable when solving large problem instances as they usually appear in practical cases [4].

2.1. Schedule Generation Schemes

Schedule generation schemes are the core of most of the heuristic/metaheuristic solution procedures for the RCPSP [4]. A SGS is a constructive technique that builds a feasible schedule by stepwise extension of a partial schedule (i.e. a schedule where only a subset of the activities have been scheduled). There are two different types of SGS: serial SGS (S-SGS) and parallel SGS (P-SGS). The S-SGS is an *activity oriented* SGS that performs activity incrementation while building the schedule. The P-SGS, is a *time oriented* SGS that performs time incrementation in the schedule build. For a formal and detailed definition of the SGS the reader is referred to [4].

3. The PSPSolver Library

The main motivation for the development of our PSPSolver is the lack of freely available software for the RCPSP. The main goal is to build an extensible environment for abstracting RCPSP instances and solutions with the implementation of SGS-based solving methods by using modern programming concepts. PSPSolver provides an extensible object-oriented application programming interface (API) for the visualization, representation, and solving of RCPSP instances. The library is currently implemented in C# and can be freely downloaded from *http://www.planningforce.com/wiki/.* The code distribution includes detailed documentation and ready-to-use code snippets. The reader is referred to this documentation for a detailed description of PSPSolver's features (e.g. easy of use, extensibility, performance, limitations, comparison with other software, etc.).

3.1. Problem and Solution Representation

The library provides classes to represent single mode RCPSP instances and schedules, nevertheless, other PSP models could be easily extended as well (i.e. multi-mode RCPSP). PSPSolver provides mechanisms for handling RCPSP logical instances by supporting common file formats (i.e. the formats proposed by PSPLIB[5]), and also defines a new normalized XML-based representation, best suitable for data exchange between applications. The library can model and be extended with user defined priority rules as well.

3.2. Visualization

PSPSolver is able to render RCPSP instances as A-O-N digraphs and also instance solutions as Gant Charts. In order to implement a clear visual representation of the instance, the network rendering relies on features of the GraphViz graph rendering engine [6]. This feature is of great utility especially when we want to visualize a complex topology on large RCPSP instances. The diagrams illustrated in Figure 1 and Figure 2 were rendered by using the PSPSolver visualization API.

3.3. Solving

PSPSolver provides an API for solving RCPSP instances by implementing the S-SGS and the P-SGS applying user defined heuristics (priority rules). Additionally, one of the most important features in the Solving API is the possibility to easily integrate or be integrated in custom scheduling metaheuristics (e.g. ACO, TS, PSO, SA, etc.). In brief, PSPSolver is able to solve a PSP using user-defined heuristics (priority rules and/or metaheuristics) by implementing the SGS approach. The library code distribution includes a self-contained example (the PSPViewer application) that illustrates the main features of the PSPSolver API by implementing an instance renderer, a solution benchmarker, a solver, an illustrated custom priority rule, and a solution renderer. As a reference, PSPViewer was able to solve the 480 instances from the J30-SM set (PSPLIB) in less than 2s. (an average of 4ms. per instance), using a S-SGS and the SPT (Shortest Processing Time) heuristic in a Pentium 2.0 GHz with 1MB of RAM using Visual C# Express 2005 and .Net Framework 2.0.

4. Conclusions

We consider that PSPSolver is a basic but powerful free library for solving the RCPSP model, and we consider it a valuable and practical tool for the PSP research community. The library can be easily adapted to the researcher's needs, in order to implement new SGS-based heuristics. We plan to improve the library with the implementation of a lower-bound calculation, double justification (schedule optimization)[7] and extending it to the multi-mode RCPSP. We are currently working in porting the library to the JAVA programming language as well.

References

[1] Brucker, P., Drexl, A., Mohring, R., Neumann, K., and Pesh, E., 1999. *Resource-constrained project scheduling: Notation, classification, models, and methods*, European Journal of Operational Research 112

[2] Herroelen, W., De Reyck, B., and Demeulemeester, E., 1998. *Resource-constrained project scheduling: a survey of recent developments*, Computers Ops Res. Vol. 25

[3] Blazewicz, J., Cellary W., Slowinski R., and Weglarz J., 1986. *Scheduling under Resource Constraints: Deterministic Models*, in: *Annals of operations research, vol. 7.*, J.C. Baltzer

[4] Kolisch, R. and Hartmann, S. eds. 1999. *Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis*, in: *Project scheduling: Recent models, algorithms and applications*, Kluwer

[5] Kolisch, R., and Sprecher, A. 1997. *PSPLIB A Project Scheduling Problem Library*, European Journal of Operational Research 96

[6] Gansner, E., and E., North, 2000. *An open graph visualization system and its applications to software engineering*, Software Practice and Experience Vol.30, 11

[7] Valls, V., Ballestín, F., Quintanilla, S., 2006. *Justification Technique Generalizations,* in: *Perspectives in Modern Project Scheduling*, Kluwer

CHARLES UNIVERSITY PRAGUE

faculty of mathematics and physics

ISSN 1368-5708