Planning & Scheduling

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Planning as SAT and CSP

Introduction

So far, we discussed **ad-hoc planning algorithms**, i.e., algorithms designed specifically to do (only) planning.

Would it possible to **convert** the planning problem to a problem **in another formalism** and to exploit existing solving techniques for that formalism?

Yes, we can use for example **SAT and CSP**.

Why?

Improvement of the SAT/CSP solver will immediately give an improvement of the planner (without an extra effort).

Problem:

SAT and CSP use **static encodings** (with a known number of variables), but we **do not know the size of plan**!

Solution:

Let us look for plans of a given length
and if no plan exists, then extend the expected plan length.

Planning as SAT – the core idea

1. The restricted-length planning problem is **encoded as a propositional formula.**
   - How to encode the states?
   - How to encode state transitions (via actions)?
2. Using a SAT solver we **find a model** of the formula (assignment of values to variables)**.**
   - Davis-Putnam, GSAT, …
3. From the model we **decode the solution** of the original problem (the plan).

Planning as SAT – state encoding

- **States** in a classical representation are **sets** (conjunction) of **instantiated atoms**.

  Example: at(r1,l1) ∧ ¬loaded(r1)
- We can directly encode **atoms** as **propositional variables**.

  Example: the state above has the following model
  {at(r1,l1)←true, loaded(r1)←false}
  - **Isn't it possible that a robot is at two places at the same time?**

    For example, if having a propositional variable for at(r1,l2), we get two models of the above formula:
    - {at(r1,l1)←true, loaded(r1)←false, at(r1,l2)←true}
    - {at(r1,l1)←true, loaded(r1)←false, at(r1,l2)←false}

    but only the second model corresponds to reality.
  - **We need to encode the implicit information too!**

    i.e. at(r1,l1) ∧ ¬at(r1,l2) ∧ ¬loaded(r1)

  Note:

  A single formula can encode a set of states together:

  ((at(r1,l1) ∧ ¬at(r1,l2)) ∨ (¬at(r1,l1) ∧ at(r1,l2))) ∧ ¬loaded(r1)

Planning as SAT – transition encoding

- The formula encoding a state does not cover dynamics of the system (state changes)!
- **The state transition** using action move(r1,l1,l2) can be described using two formulas:
  - at(r1,l1) ∧ ¬at(r1,l2) % the state before the transition
  - ¬at(r1,l1) ∧ at(r1,l2) % the state after the transition
- **Different atoms hold at different states!**
- We extend the attributes of atoms with the name of the state (**fluents**) so the transition can be encoded using a single **formula**:
  - at(r1,l1,s1) ∧ ¬at(r1,l2,s1) ∧ ¬at(r1,l1,s2) ∧ at(r1,l2,s2)
- Plus, we add a new propositional variable describing the action that caused the transition:
  - move(r1,l1,l2,s1) ∧ at(r1,l1,s1) ∧ ¬at(r1,l2,s1) ∧ ¬at(r1,l1,s2) ∧ at(r1,l2,s2)
- But how do we know that action *move* is applied to state *s1*?

Planning as SAT – problem encoding

We are looking for **plans with a given maximal length** (n).

For each step *i* of the plan we have:

- **variables** (**fluents**) L describing the atoms in the **state**, such as at(r1,l1,i),
- **variables** describing possible **actions** A, such as move(r1,l1,l2,i).

The planning problem is encoded as a conjunction of the following formulas:

- Formula encoding the **initial state** (a compete description of the state):
  $\wedge\{p_0 \mid p \in s_0\} \wedge \wedge\{\neg p_0 \mid p \in L - s_0\}$
- Formula encoding the goal (a partial encoding of the state):
  $\wedge\{p \mid p_n \in g^+\} \wedge \wedge\{\neg p_n \mid p \in g^-\}$
- For each action we have a formula describing the state transition caused by that action in step *i* of the plan:
  - $a_i \Rightarrow \wedge\{p_i \mid p \in \text{precond}(a)\} \wedge \wedge\{e_{i+1} \mid e \in \text{effects}(a)\}$
- **Axiom of complete exclusion**:
  - No two different actions *a* and *b* can be applied simultaneously in any step:
    $\neg a_i \vee \neg b_i$

Is it enough?

Planning as SAT – frame axioms

- We need to ensure that atoms, which are not modified by a selected action, remain the same in the next state.

- **Frame axioms**:
  formulas describing what is not changed between the states
  there are several ways how to encode them:
  - **classical frame axioms**
    - each action describes which atoms p are not changed by the action ($p \notin effects(a)$)
      $a_i \Rightarrow (p_i \Leftrightarrow p_{i+1})$
    - we also need to ensure that some action is selected in each step
      $V\{a_i \mid a \in A\}$
  - **frame axioms with explanation**
    - the truth value of the fluent is changed if only if some action changes it
      $(\neg p_i \wedge p_{i+1} \Rightarrow V\{a_i \mid p \in effects^+(a)\})$;; some action added p
      $\wedge\ (p_i \wedge \neg p_{i+1} \Rightarrow V\{a_i \mid p \in effects^-(a)\})$;; some action deleted p
    - we need an **axiom of complete exclusion** there (that leads to linearly ordered plans only)
    - it is enough to use an **axiom of conflicting exclusion**, i.e., two dependent actions cannot be selected in a single step (recall the planning graph)

Planning as SAT – alternative action encodings

- Let us now try to decrease the number of variables necessary to encode the actions.
  (less number of variables = smaller search space)
- Situation: 3 robots and 3 locations
  - To model operator move(r,loc1,loc2) we need 3×3×3= **27 variables** in each layer.
  - The variable for action move(r1,loc1,loc2) can be substituted by a conjunction of three variables describing the action attributes:
    move1(r1) ∧ move2(loc1) ∧ move3(loc2)
  - Now, several actions of the same operator share variables (for example move2(loc1) is used for move(r1,loc1,loc2) as well as for move(r2,loc1,loc3)) so we have 3+3+3= **9 variables** in each layer.
- **We can even share variables between actions of different (but similar) operators**, for example move and fly.

Planning as SAT – final notes

- **The framework for planning as SAT**

  for n = 0, 1, 2, …,

  encode the restricted-length planning problem (P,n) as SAT Φ

  if Φ is satisfiable then

  from the model of Φ extract the solution plan

  – **Plan extraction**
    - in each step I there is (exactly) one variable a_i with the value *true*, that determines the action in a plan

- **How does it work?**
  – **not very practical in its core form** (time and memory consumption)
  – much better in **combination** with other ideas such as the **planning graph**
    - system **BlackBox** encodes the planning graph as SAT using variables for actions and atoms in the graph
    - satisfiability of the formula is used for plan extraction

Planning as SAT – example

- **Planning domain:**
  – one robot: r1
  – two connected locations: l1, l2
  – one operator: move
- **Encoding of the planning problem** (P,n) for plans of length n = 1
  – initial state: {at(r1,l1)}
  at(r1,l1,0) ∧ ¬at(r1,l2,0)
  – goal: {at(r1,l2)}
  at(r1,l2,1)
  – operator: move(r,l,l') = (precond: at(r,l), effects: at(r,l'), ¬at(r,l))
  move(r1,l1,l2,0) ⇒ at(r1,l1,0) ∧ at(r1,l2,1) ∧ ¬at(r1,l1,1)
  move(r1,l2,l1,0) ⇒ at(r1,l2,0) ∧ at(r1,l1,1) ∧ ¬at(r1,l2,1)
  – axiom of complete exclusion:
  ¬move(r1,l1,l2,0) ∨ ¬move(r1,l2,l1,0)
  – frame axioms with explanation:
  ¬at(r1,l1,0) ∧ at(r1,l1,1) ⇒ move(r1,l2,l1,0)
  ¬at(r1,l2,0) ∧ at(r1,l2,1) ⇒ move(r1,l1,l2,0)
  at(r1,l1,0) ∧ ¬at(r1,l1,1) ⇒ move(r1,l1,l2,0)
  at(r1,l2,0) ∧ ¬at(r1,l2,1) ⇒ move(r1,l2,l1,0)

- **Planning as a CSP**
  - planning problem is encoded fully as a CSP
    - SAT is a special case of a CSP
    - but we will use a novel representation of planning problems – a **multi-valued state-variable representation**
- **A CSP in planning**
  - constraint satisfaction techniques can be used in planning algorithms (for example in Graphplan to extract a plan)
  - this is currently a more common way of exploiting a CSP in planning

- **Multi-valued state variables** describe the properties of objects that are changing between the states (by actions).
  - rloc: robots × S → locations
  - rload: robots × S → containers ∪ {nil}
  - cpos: containers × S → locations ∪ robots
- **Rigid relations** are (still) represented using relations.
  - adjacent(loc1,loc2)
  - robots(r1) ;; describes the types of constants
- **Operators** describe changes of state variables.
  - **move(r,l,m)**
    ;; robot r at location l moves to an adjacent position m
    precond: rloc(r)=l, adjacent(l,m)
    effects: rloc(r)←m
  - **load(c,r,l)**
    ;; robot r loads container c at location l
    precond: rloc(r)=l, cpos(c)=l, rload(r)=nil
    effects: rload(r)←c, cpos(c)←r
  - **unload(c,r,l)**
    ;; robot r unloads container c at location
    precond: rloc(r)=l, rload(r)=c
    effects: rload(r)←nil, cpos(c)←l

Planning as a CSP – variables

- Again, we model a restricted-length planning problem, i.e., we are looking for **plans of maximal length k**.
- For each state variable x_i we have CSP variables $x_i(j)$ $0 \leq j \leq k$ with the domain describing possible values of the state variable.
  - **Example**:
    - rloc(r1,j)$\in${l1,l2,l3}, $0 \leq j \leq 4$
    - rload(r1,j)$\in${c1,c2,c3,nil}, $0 \leq j \leq 4$
    - cpos(c,j)$\in${l1,l2,l3,r1}, $0 \leq j \leq 4$, c$\in${c1,c2,c3}
- For each layer j, $0 \leq j \leq k-1$, we have a single **action variable** act(j) with the domain describing possible actions (including no-op).
  - **Example**:
    - act(j) $\in${move(r1,l1,l2),move(r1,l1,l3),…, no-op}, $0 \leq j \leq 3$

Planning as a CSP – state encoding

- **The initial and goal states** are encoded using **unary constraints** (assignment of states variables):
  - if the state variable x_i in the initial state s_0 has a value v_i, then add constraint:
    - $x_i(0) = v_i$

    Example:
    rloc(r1,0)=l1, rload(r1,0)=nil, cpos(c1,0)=l1, cpos(c2,0)=l2, cpos(c3,0)=l2
  - if the state variable x_i in the goal state g has a value v_i, then add constraint:
    - $x_i(k) = v_i$

    Example:
    cpos(c1,4)=l2, cpos(c2,4)=l1

Planning as a CSP – action encoding

- Actions are encoded as constraints connecting an action variable act(j) with neighbouring state variables $x_i(j)$ and $x_i(j+1)$.
- **Action precondition** in the form **$(x_i=v_i)$** is encoded as constraints $act(j)=a \Rightarrow x_i(j)=v_i$, $0 \leq j \leq k-1$.
- **Action precondition** in the form **$(x_i \in D_i)$** is encoded as constraints $act(j)=a \Rightarrow x_i(j) \in D_i$, $0 \leq j \leq k-1$.
- Assignment **$x_i \leftarrow v_i$** from **action effects** is encoded as constraints $act(j)=a \Rightarrow x_i(j+1)=v_i$, $0 \leq j \leq k-1$.

Example:

- act(j)=move(r1,l1,l2) ⇒ rloc(r1,j)=l1 ;; precondition
- act(j)=move(r1,l1,l2) ⇒ rloc(r1,j+1)=l2 ;; effect

Note: another precondition adjacent(l1,l2), that is rigid, has been assumed when grounding the action from the operator (selecting the action attributes)

Planning as a CSP – frame axioms

- Similarly to SAT we need to ensure that state variables, that are not changed by the selected action, preserve their value between the states – **frame axioms**.
- We can use **ternary constraints** connecting the action variable act(j) with the neighbouring state variables $x_i(j)$ and $x_i(j+1)$.
- In particular, if the state variable x_i is not among the effects of action *a* (x_i is **invariant** with respect to *a*), then add the following constraints:
  $act(j)=a \Rightarrow x_i(j)=x_i(j+1)$, $0 \leq j \leq k-1$.

Example:

- act(j)=move(r1,l1,l2) ⇒ rload(r1,j)=rload(r1,j+1)
- act(j)=move(r1,l1,l2) ⇒ cpos(c,j)=cpos(c,j+1), c∈{c1,c2,c3}

CSP for planning

- **Constraint satisfaction techniques can be used as a part of planning algorithms.**
- plan-space planning
  - verifying consistency of a partial plan using constraint propagation – arc consistency (fast but incomplete)
  - complete consistency test when the flaw-less plan is found
- **planning-graph planning**
  - planning graph is a static structure that can be easily encoded as a CSP
  - constraint satisfaction techniques are used to extract the plan from the planning graph

Do & Kambhampati (2000)

Graphplan as a CSP

- We will use variables for atoms, where the values will identify the actions that make the atoms true.

CSP model:

- **variables**
  - nodes $P_{j,m}$ from the state layers (atom p_j at layer m)
  - we index only the state layers (not the action layers)
- **variables' domains**
  - actions that have a given atom among its effects
  - $\perp$ for atom that is not true
- **constraints**
  - connect positive effects of actions with preconditions
  - mutex relations

Graphplan as a CSP – constraints

$P_{4,m}=a \Rightarrow P_{1,m-1}\neq\bot \wedge P_{2,m-1}\neq\bot \wedge P_{3,m-1}\neq\bot$

- action a has preconditions p_1, p_2, p_3 and positive effect p_4
- this constraint is used for all positive effects of action a

$P_{i,m}=\bot \vee P_{j,m}=\bot$

- mutex between atoms p_i and p_j

$P_{i,m}\neq a \vee P_{j,m}\neq b$

- actions a, b are mutex and p_i is a positive effect of a and p_j is a positive effect of b

$P_{i,k}\neq\bot$

- p_i is a goal atom and k is the index of the last layer

no parallel actions

- at most one action different from no-op is assigned to variables in the same layer

no empty layers

- at least one action different from no-op is assigned to variables in the same layer

Graphplan as a Boolean CSP

- We will use Boolean variables for atoms and actions, where value true means valid atom and used action.

Boolean CSP (SAT) model:

- **variables**
  - Boolean variables for actions $A_{j,m}$ and for atoms $P_{j,n}$
  - layers are indexed as in the planning graph, state layers starting with zero, action layers starting with one
- **variables' domains**
  - value **true** means that atom is valid and action is selected
- **constraints**
  - connect action variables with preconditions and positive effects
  - mutex constraints

Graphplan as a Boolean CSP – constraints

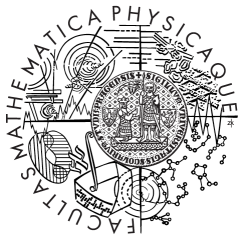
- **constraints for preconditions**
  - $\mathbf{A_{i,m} \Rightarrow P_{j,m-1}}$
  - p_j is a precondition of action a_i
- **successor state constraints**
  - $\mathbf{P_{i,m} \Leftrightarrow (\vee_{p_i \in effects^+(a_j)} A_{j,m}) \vee (P_{i,m-1} \wedge (\wedge_{p_i \in effects^-(a_j)} \neg A_{j,m}))}$
  - atom p_i is active in the layer, if the atom is either added by some action or the atom was true in the previous layer and no action deleted it
  - no-op is not used there!
  - Beware! These constraints allow the same atom to be added by one action and deleted by another action so we need mutex constraints to exclude simultaneous usage of such actions in the same layer!
- **mutex constraints**
  - $\mathbf{\neg A_{i,m} \vee \neg A_{j,m}}$ mutex of actions a_i and a_j at layer m
  - $\mathbf{\neg P_{i,n} \vee \neg P_{j,n}}$ mutex of atoms p_i and p_j at layer n
- **initial state and goal**
  - $\mathbf{P_{i,k}=true}$ p_i is a goal atom and k is the index of the last state layer
  - $\mathbf{P_{i,0}=true}$ p_i is an atom from the initial state
- **other constraints**
  - no parallel actions – at most one action is active in each layer
  - no empty layers – at least one action is active in each layer

SAT vs. CSP

- **Both SAT and CSP-based planning exploit the same core principle.**
- **CSP encoding is more compact**
  - a simpler representation of states thanks to multi-valued state variables (gives less variables in total)
  - no action mutex constraints are necessary (for sequential plans)

Note:

- CSP and SAT problems are NP-c while planning is PSPACE- or NEXPTIME-c. How is it possible?
- The encoding of planning problems as SAT and a CSP brings **exponential increase of the problem size**.
  - SAT: exponential number of Boolean variables
  - CSP: linear number of variables, but exponential size of a CSP due to variables' domains



Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz