Foundations of constraint satisfaction

Roman Barták
Charles University in Prague

bartak@ktiml.mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

What is the course about?

Constraint satisfaction problems

Algorithms for solving constraint satisfaction problems

- Local search
  - HC, MC, RW, Tabu Search
- Search algorithms
  - GT, BT, BJ, BM, DB, LDS
- Consistency techniques
  - NC, AC, DAC, PC, DPC, RPC, SC
- Search and constraint propagation
  - FC, PLA, LA
- Optimisation problems
  - B&B
- Over-constrained problems
  - PCSP, constraint hierarchies

What is a constraint?

Constraint is an arbitrary relation over the set of variables.

- every variable has a set of possible values - a domain
  - this course covers discrete finite domains only
- the constraint restricts the possible combinations of values

Some examples:

- the circle C is inside a square S
- the length of the word W is 10 characters
- X is less than Y
- a sum of angles in the triangle is 180°
- the temperature in the warehouse must be in the range 0-5°C
- John can attend the lecture on Wednesday after 14:00

Constraint can be described:

- intentionally (as a mathematical/logical formula)
- extensionally (as a table describing compatible tuples)

Constraint Satisfaction Problem

CSP (Constraint Satisfaction Problem) consists of:

- a finite set of variables
- domains - a finite set of values for each variable
- a finite set of constraints

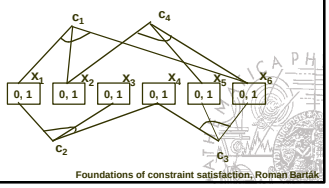
A solution to CSP is a complete assignment of variables satisfying all the constraints.

CSP is often represented as a (hyper)graph.

Example:

variables $x_1,\ldots,x_6$
domain {0,1}

c_1: $x_1+x_2+x_6=1$
c_2: $x_1-x_3+x_4=1$
c_3: $x_4+x_5-x_6>0$
c_4: $x_2+x_5-x_6=0$



A bit of history

Artificial Intelligence
Scene labelling (Waltz 1975)

Interactive graphics
Sketchpad (Sutherland 1963)
ThingLab (Borning 1981)

Logic programming
unification ® constraint solving (Gallaire 1985, Jaffar, Lassez 1987)

Operations research and discrete mathematics
NP-hard combinatorial problems

Some toy problems

SEND + MORE = MONEY

assign different numerals to different letters
S and M are not zero

A constraint model (with a carry bit):

```
E,N,D,O,R,Y in 0..9, S,M in 1..9, P1,P2,P3::0..1
all_different(S,E,N,D,M,O,R,Y)
    D+E = 10*P1+Y
  P1+N+R = 10*P2+E
  P2+E+O = 10*P3+N
  P3+S+M = 10*M +O
```

N-queens problem

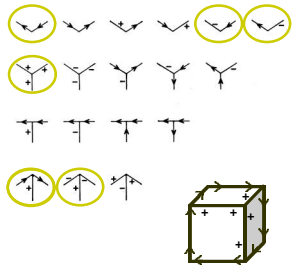
allocate N queens to the chessboard
the queens do not attack each other

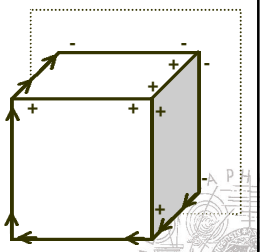
A constraint model:

queens in columns "i r(i) in 1..N
no conflict
"i¹j r(i)¹r(j) & |i-j|¹|r(i)-r(j)|

Constraints in scene labelling (Waltz 1975)

Looking for feasible interpretation of 3D lines in 2D drawing
First usage of constraint propagation techniques

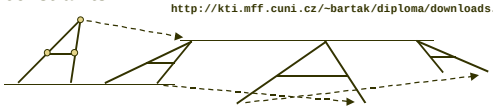


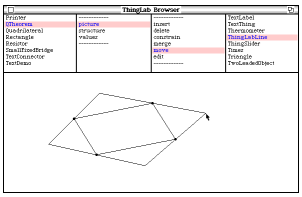


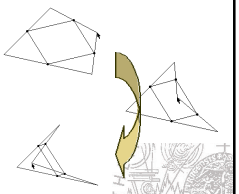
Constraints in interactive graphics

How to manipulate a graphical object described by constraints?

http://kti.mff.cuni.cz/~bartak/diploma/downloads.html



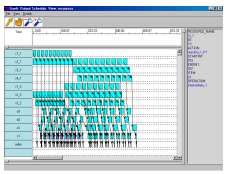


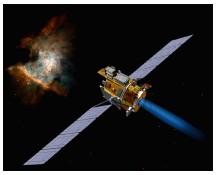


http://www.cs.washington.edu/research/constraints/

Constraints in A.I. planning and scheduling

Scheduling problem =
a set of activities has to be processed by a limited number of resources in a limited amount of time.
Combinatorial optimisation



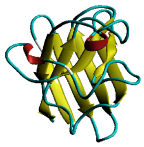


Planning problem =
find a set of activities to achieve a given goal
Deep Space One
– autonomous planning of spacecraft activities

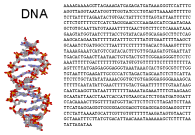
Constraints in bioinformatics

http://www.soi.city.ac.uk/~drg/bioinformatics/

Design of a 3D protein structure from the sequence of amino-acids (3D structure determines features of proteins)



Analysing a sequence of DNA, estimating a distance between DNAs, comparing DNAs



Solving constraints by enumeration

Constraints are used only as a test
assign values to variables ...
... and see what happens

systematic search
explores the space of all assignments systematically
GT, BT, BJ, BM, DB, LDS

non-systematic search
some assignments may be skipped during search
Credit Search, Bounded Backtrack

local search
explore the search space by small steps
HC, MC, RW, Tabu, *GSAT, Genet, simulated annealing*

Systematic search

Explore systematically the space of all assignments
systematic = every valuation will be explored sometime

Features:
+ complete (if there is a solution, the method finds it)
- it could take a lot of time to find the solution

Basic classification:
Explore complete assignments

generate and test
such search space is used by local search (non-systematic)

Extending partial assignments
tree search

Generate and test (GT)

The most general problem solving method

1) generate a candidate for solution
2) test if the candidate is really a solution

How to apply GT to CSP?

1) assign values to all variables
2) test whether all the constraints are satisfied

GT explores complete but inconsistent assignments until a (complete) consistent assignment is found.

Procedure GT(X:variables, C:constraints)

V ¬ construct a first complete assignment of X
while V does not satisfy all the constraints C do
  V ¬ construct systematically a complete assignment next to V
end while
return V

Weaknesses and improvements of GT

The greatest weakness of GT is exploring too many "visibly" wrong assignments.

Example:
X,Y,Z::{1,2} X = Y, X ¹ Z, Y > Z

X	1	1	1	1	2	2	2
Y	1	1	2	2	1	1	2
Z	1	2	1	2	1	2	1

How to improve generate and test?

smart generator

smart (perhaps non-systematic) generator that uses result of test
  å local search techniques

earlier detection of clash

constraints are tested as soon as the involved variables are instantiated ® backtracking-based search

Local search

Generate and test **explores complete but inconsistent assignments until a complete consistent assignment is found.**

Weakness of GT - the generator does not use result of test

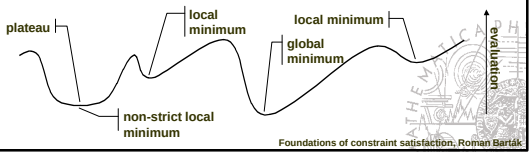
The next assignment can be constructed in such a way that constraint violation is smaller.

- **only "small" changes of the assignment are allowed**
- **next assignment should be "better" than previous**
  better = more constraints are satisfied
- **assignments are not necessarily generated systematically**
  we lost completeness but we (hopefully) get better efficiency

Local search **is a technique of searching solution by small changes (local steps) to the solution candidate.**

Local search - Terminology

state - a complete assignment of values to variables
evaluation - a value of the objective function (# violated constraints)
neighbourhood - a set of states locally different from the current state (the states differ from the current state in the value of one variable)
local optimum - a state that is not optimal and there is no state with better evaluation in its neighbourhood
strict local optimum - a state that is not optimal and there are only states with worse evaluation in its neighbourhood
non-strict local optimum - local optimum that is not strict
global optimum - the state with the best evaluation
plateau - a set of neighbouring states with the same evaluation



Hill Climbing

Hill climbing is perhaps the most known technique of local search.
  start at **randomly generated state**
  look for **the best state in the neighbourhood** of the current state
    neighbourhood = differs in the value of any variable
    neighbourhood size = $S_{i=1..n}(D_i-1)$ (= n*(d-1))
  "escape" from the local optimum via **restart**

Algorithm Hill Climbing

```
procedure hill-climbing(Max_Flips)
    restart: s ¬ random assignment of variables;
    for j:=1 to Max_Flips do            % restricted number of steps
        if eval(s)=0 then return s
        if s is a strict local minimum then
            go to restart
        else
            s ¬ neighbourhood with the smallest evaluation value
        end if
    end for
    go to restart
end hill-climbing
```


Min-Conflicts (Minton, Johnston, Laird 1997)

Observation:

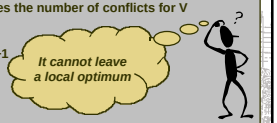
- **the hill climbing neighbourhood is pretty large (n*(d-1))**
- **only change of a conflicting variable may improve the valuation**

Min-conflicts method

select **randomly a varible in conflict** and try to **improve it**
  neighbourhood = different values for the selected variable *i*
  neighbourhood size = (D_i-1) (= (d-1))

Algorithm Min-Conflicts

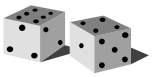
```
procedure MC(Max_Moves)
    s ¬ random assignment of variables
    nb_moves ¬ 0
    while eval(s)>0 & nb_moves<Max_Moves do
        choose randomly a variable V in conflict
        choose a value v' that minimises the number of conflicts for V
        if v' ¹ current value of V then
            assign v' to V
            nb_moves ¬ nb_moves+1
        end if
    end while
    return s
end MC
```



Random walk

How to leave the local optimum without a restart (i.e. via a local step)?

By adding some “noise” to the algorithm!



Random walk

- a state from the neighbourhood is selected randomly (e.g., the value is chosen randomly)
- such technique can hardly find a solution
- so it needs some guide

Random walk can be combined with the heuristic guiding the search via probability distribution:

- *p* - probability of using the random walk
- *(1-p)* - probability of using the heuristic guide

Min-Conflicts Random Walk

MC guides the search (i.e. satisfaction of all the constraints) and RW allows us to leave the local optima.

Algorithm Min-Conflicts-Random-Walk

```
procedure MCRW(Max_Moves,p)
    s ¬ random assignment of variables
    nb_moves ¬ 0
    while eval(s)>0 & nb_moves<Max_Moves do
        if probability p verified then
            choose randomly a variable V in conflict
            choose randomly a value v' for V
        else
            choose randomly a variable V in conflict
            choose a value v' that minimises the number of conflicts for V
        end if
        if v' ¹ current value of V then
            assign v' to V
            nb_moves ¬ nb_moves+1
        end if
    end while
    return s
end MCRW
```

0.02 £ p £ 0.1

Steepest Descent Random Walk

Random walk can be combined with the hill climbing heuristic too.

Then, no restart is necessary.

Algorithm Steepest-Descent-Random-Walk

```
procedure SDRW(Max_Moves,p)
    s ¬ random assignment of variables
    nb_moves ¬ 0
    while eval(s)>0 & nb_moves<Max_Moves do
        if probability p verified then
            choose randomly a variable V in conflict
            choose randomly a value v' for V
        else
            choose a move <V,v'> with the best performance
        end if
        if v' ¹ current value of V then
            assign v' to V
            nb_moves ¬ nb_moves+1
        end if
    end while
    return s
end SDRW
```

Tabu list

Observation:

Being trapped in local optimum is a special case of cycling.

How to avoid cycles in general?

Remember already visited states and do not visit them again.

- memory consuming (too many states)

It is possible to remember just few last states.

- prevents „short“ cycles

Tabu list = a list of forbidden states

the state can be represented by a selected attribute

á variable, value ñ - describes the change of the state (a previous value)

tabu list has a fix length *k* (tabu tenure)

„old“ states are removed from the list when a new state is added

state included in the tabu list is forbidden (it is tabu)

Aspiration criterion = enabling states that are tabu

i.e., it is possible to visit the state even if the state is tabu

example: the state is better than any state visited so far

Tabu search (Galinier, Hao 1997)

The tabu list prevents short cycles.

It allows only the moves out of the tabu list or the moves satisfying the aspiration criterion.

Algorithm Tabu Search

```
procedure tabu-search(Max_Iter)
    s ¬ random assignment of variables
    nb_iter ¬ 0
    initialise randomly the tabu list
    while eval(s)>0 & nb_iter<Max_Iter do
        choose a move <V,v'> with the best performance among the non-tabu
            moves and the moves satisfying the aspiration criteria
        introduce <V,v> in the tabu list, where v is the current value of V
        remove the oldest move from the tabu list
        assign v' to V
        nb_iter ¬ nb_iter+1
    end while
    return s
end tabu-search
```


Localizer (Michel, Van Hentenryck 1997)

The local search algorithms have a similar structure that can be encoded in the common skeleton. This skeleton is filled by procedures implementing a particular technique.

Local Search Skeleton

```
procedure local-search(Max_Tries,Max_Moves)
    s ¬ random assignment of variables
    for i:=1 to Max_Tries while Gcondition do
        for j:=1 to Max_Moves while Lcondition do
            if eval(s)=0 then
                return s
            end if
            select n in neighbourhood(s)
            if acceptable(n) then
                s ¬ n
            end if
        end for
        s ¬ restartState(s)
    end for
    return best s
end local-search
```