Foundations of constraint satisfaction

2

Roman Barták
Charles University in Prague

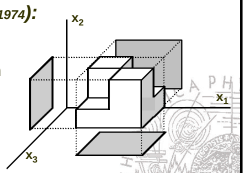
bartak@ktiml.mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Binary constraints

World is not binary …
but it could be transformed to a binary one!

Each CSP can be transformed to an equivalent binary CSP
- many CSP algorithms designed for binary problems
- still open efficiency issues

Projection technique (Montanary 1974):
- straightforward but
- does not give an equivalent problem
- bound consistency
  - better efficiency
  - weaker pruning



Dual encoding

Swapping variables and constraints.

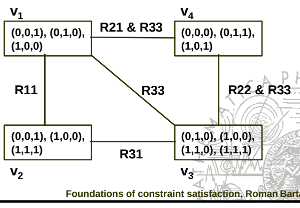
k- ary constraint c is converted to
a dual variable v_c with the domain consisting of compatible tuples

for each pair of constraints c a c' sharing some variables there is
a binary constraint between v_c a $v_{c'}$ restricting the dual variables
to tuples in which the original shared variables take the same value

Example:

variables $x_1,…,x_6$
with 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: $x_2+x_5-x_6=0$



Hidden variable encoding

New dual variables for (non-binary) constraints.

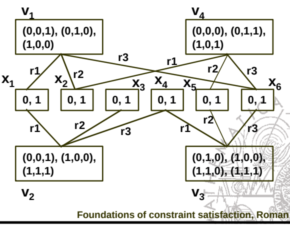
k- ary constraint c is translated to
a dual variable v_c with the domain consisting of compatible tuples

for each variable x in the constraint c there is a constraint between
x a v_c restricting tuples of dual variable to be compatible with x

Example:

variables $x_1,…,x_6$
with domains {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: $x_2+x_5-x_6=0$

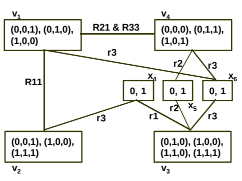


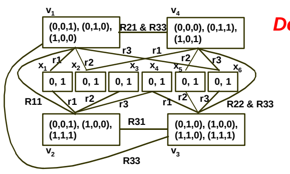
Other encodings

Hybrid encoding

transformation between dual and
hidden variable encoding

contains parts of both encodings





Double encoding

hidden and original variables
are included

constraints from both
encodings are used

improved propagation

Backtracking

Probably the most widely used systematic search algorithm
basically it is depth-first search

Using backtracking to solve CSP
1) assign values gradually to variables
2) after each assignment test the constraints over the assigned variables (and backtrack upon failure)

Extends a partial consistent assignment until a complete consistent assignment is found.

Open questions:

what is the order of variables?
- variables with a smaller domain first
- variables participating in more constraints first
- "key" variables first

what is the order of values?
- problem dependent

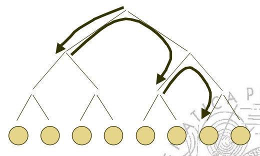
Algorithm chronological backtracking

A recursive definition

Algorithm BT(X:variables, V:assignment, C:constraints)

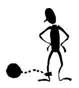
```
if X={} then return V
x ¬ select a not-yet assigned variable from X
for each value h from the domain of x do
    if constraints C are satisfied over V+x/h then
        R ¬ BT(X-x, V+x/h, C)
        if R¹fail then return R
end for
return fail
```

top call BT(X, {}, C)



Backtracking is always better than generate and test!

Weaknesses of backtracking



thrashing

throws away the reason of the conflict

Example: A,B,C,D,E:: 1..10, A>E

BT tries all the assignments for B,C,D before finding that A¹1

Solution: backjumping (jump to the source of the failure)

redundant work

unnecessary constraint checks are repeated

Example: A,B,C,D,E:: 1..10, B+8<D, C=5*E

when labelling C,E the values 1,..,9 are repeatedly checked for D

Solution: backmarking, backchecking (remember (no-)good assignments)

late detection of the conflict

constraint violation is discovered only when the values are known

Example: A,B,C,D,E::1..10, A=3*E

the fact that A>2 is discovered when labelling E

Solution: forward checking (forward check of constraints)

Backjumping (Gaschnig 1979)

Backjumping is used to remove thrashing.

How?

1) identify the source of the conflict (impossible to assign a value)
2) jump to the past variable in conflict

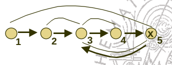
The same run like in backtracking, only the back-jump can be longer, i.e. irrelevant assignments are skipped!

How to find a jump position? What is the source of the conflict?

select the constraints containing just the currently assigned variable and the past variables

select the closest variable participating in the selected constraints

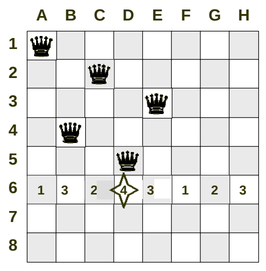
Graph-directed backjumping



Enhancement: use only the violated constraints

Conflict-directed backjumping in practice

N-queens problem



Queens in rows are allocated to columns.

6th queen cannot be allocated!

1. Write a number of conflicting queens to each position.

2. Select the farthest conflicting queen for each position.

3. Select the closest conflicting queen among positions.

Note:

Graph-directed backjumping has no effect here (due to complete graph)!

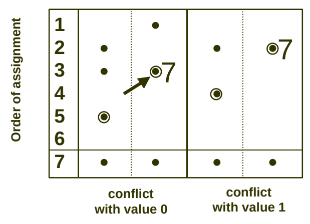
Identification of the conflicting variable

How to find out the conflicting variable?

Situation:

assume that the variable no. 7 is being assigned (values are 0, 1)

the symbol · marks the variables participating the violated constraints (two constraints for each value)



Neither 0 nor 1 can be assigned to the seventh variable!

1. Find the closest variable in each violated constraint (o).

2. Select the farthest variable from the above chosen variables for each value (7).

3. Choose the closest variable from the conflicting variables selected for each value and jump to it.

Consistency check for backjumping

In addition to the test of satisfaction of the constraints, the closest conflicting level is computed

```
procedure consistent(Labelled, Constraints, Level)
  J ¬ Level            % the level to which we will jump
  NoConflict ¬ true    % remember if there is any conflict
  for each C in Constraints do
    if all variables from C are Labelled then
      if C is not satisfied by Labelled then
        NoConflict ¬ false
        J ¬ min {J, max{L | X in C & X/V/L in Labelled & L<Level}}
      end if
    end if
  end for
  if NoConflict then return true
                else return fail(J)
end consistent
```

Algorithm backjumping

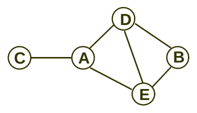
```
procedure BJ(Unlabelled, Labelled, Constraints, PreviousLevel)
    if Unlabelled = {} then return Labelled
    pick first X from Unlabelled
    Level ¬ PreviousLevel+1
    Jump ¬ 0
    for each value V from $D_X$ do
        C ¬ consistent({X/V/Level} È Labelled, Constraints, Level)
        if C = fail(J) then
            Jump ¬ max {Jump, J}
        else
            Jump ¬ PreviousLevel
            R ¬ BJ(Unlabelled-{X},{X/V/Level} È Labelled,Constraints, Level)
            if R ¹ fail(Level) then return R            % success or back-jump
        end if
    end for
    return fail(Jump)      % jump to the conflicting variable
end BJ

top call BJ(Variables,{},Constraints,0)
```

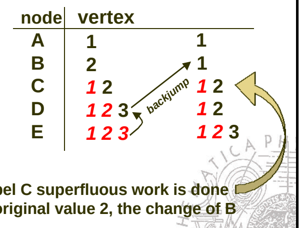
Weakness of backjumping

When jumping back the in-between assignment is lost!

Example:

colour the graph in such a way that the connected vertices have different colours



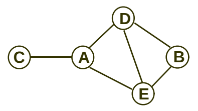


node	vertex	
A	1	1
B	2	1
C	*1* 2	*1* 2
D	*1* *2* 3	*1* 2
E	*1* *2* *3*	*1* *2* 3

During the second attempt to label C superfluous work is done - it is enough to leave there the original value 2, the change of B does not influence C.

Dynamic backtracking - example

The same graph (A,B,C,D,E), the same colours (1,2,3) but a different approach.



Backjumping
+ remember the source of the conflict
+ carry the source of the conflict
+ change the order of variables

= DYNAMIC BACKTRACKING

node	1	2	3
A	·		
B		·	
C	A	·	
D	A	B	·
E	A	B	D

jump back
+ carry the conflict source

node	1	2	3
A	·		
B		·	
C	A	·	
D	A	B	AB
E	A	B	

jump back
+ carry the conflict source
+ change the order of B, C

node	1	2	3
A	·		
C	A	·	
B	·	A	
D	A	·	
E	A	B	·

·	selected colour
AB	a source of the conflict

The vertex C (and the possible sub-graph connected to C) is not re-coloured.

Algorithm dynamic backtracking (Ginsberg 1993)

```
procedure DB(Variables, Constraints)
    Labelled ¬ {};   Unlabelled ¬ Variables
    while Unlabelled ¹ {} do
        select X in Unlabelled
        $Values_X$ ¬ $D_X$ - {values inconsistent with Labelled using Constraints}
        if $Values_X$ = {} then
            let E be an explanation of the conflict (set of conflicting variables)
            if E = {} then failure
            else
                let Y be the most recent variable in E
                unassign Y (from Labelled) with eliminating explanation E-{Y}
                remove all the explanations involving Y
            end if
        else
            select V in $Values_X$
            Unlabelled ¬ Unlabelled - {X}
            Labelled ¬ Labelled È {X/V}
        end if
    end while
    return Labelled
end DB
```

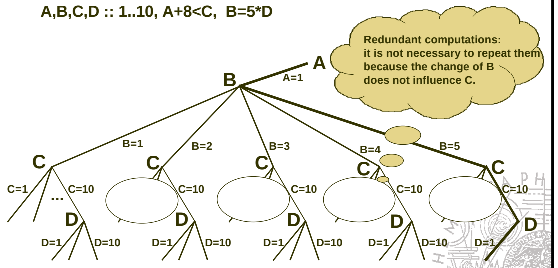
Redundant work in backtracking

What is redundant work?

repeated computation whose result has already been obtained

Example:

A,B,C,D :: 1..10, A+8<C, B=5*D

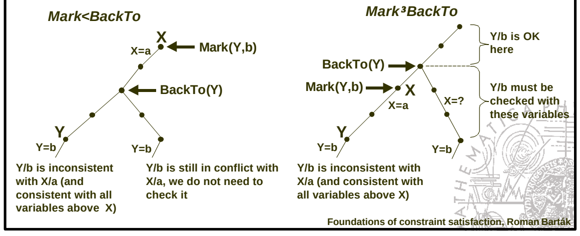


Backmarking (Haralick, Elliot 1980)

Removes redundant constraint checks by memorising negative and positive tests:

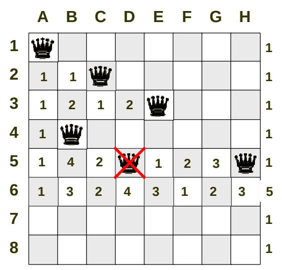
- *Mark(X,V)* is the farthest (instantiated) variable in conflict with the assignment X=V
- *BackTo(X)* is the farthest variable to which we backtracked since the last attempt to instantiate X

Now, some constraint checks can be omitted:


Mark<BackTo
X
X=a
Mark(Y,b)
BackTo(Y)
Y
Y=b
Y=b
Y/b is inconsistent with X/a (and consistent with all variables above X)
Y/b is still in conflict with X/a, we do not need to check it
Mark³BackTo
Y/b is OK here
BackTo(Y)
Mark(Y,b)
X
X=a
X=?
Y/b must be checked with these variables
Y
Y=b
Y=b
Y/b is inconsistent with X/a (and consistent with all variables above X)

Backmarking in practice

N-queens problem



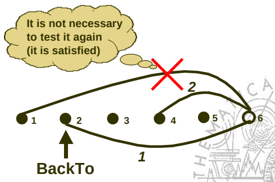
1. Queens in rows are allocated to columns.
2. Latest choice level is written next to chessboard (BackTo). At beginning 1s.
3. Farthest conflict queen at each position (MarkTo). At beginning 1s.
4. 6th queen cannot be allocated!
5. Backtrack to 5, change BackTo.
6. When allocating 6th queen, all the positions are still wrong (MarkTo<BackTo).

Note:
backmarking can be combined with backjumping (for free)

Consistency check for backmarking

Only the constraints where any value is changed are re-checked, and the farthest conflicting level is computed.

```
procedure consistent(X/V, Labelled, Constraints, Level)
    for each Y/VY/LY in Labelled such that LY ³BackTo(X) do
        % only possible changed variables Y are explored
        % in the increasing order of LY (first the oldest one)
        if X/V is not compatible with Y/VY using Constraints then
            Mark(X,V) ¬ LY
            return fail
        end if
    end for
    Mark(X,V) ¬ Level-1
    return true
end consistent
```



Algorithm backmarking

```
procedure BM(Unlabelled, Labelled, Constraints, Level)
  if Unlabelled = {} then return Labelled
  pick first X from Unlabelled        % fix order of variables
  for each value V from D_X do
    if Mark(X,V) ³ BackTo(X) then % re-check the value
      if consistent(X/V, Labelled, Constraints, Level) then
        R ¬ BM(Unlabelled-{X}, Labelled È{X/V/Level}, Constraints, Level+1)
        if R ¹ fail then return R     % solution found
      end if
    end if
  end for
  BackTo(X) ¬  Level-1                % jump will be to the previous variable
  for each Y in Unlabelled do         % tell everyone about the jump
    BackTo(Y) ¬  min {Level-1, BackTo(Y)}
  end for
  return fail        % return to the previous variable
end BM
```

Tree search and heuristics

Observation 1:
The search space for real-life problems is so huge that it cannot be fully explored.

Heuristics **- a guide of search**
- they recommend a value for assignment
- quite often leads to solution

What to do upon a failure of the heuristics?

BT cares about the end of search (a bottom part of the search tree)
- so it rather repairs later assignments than the earliest ones
- it assumes that the heuristic guides it well in the top part

Observation 2:
The heuristics are less reliable in the earlier parts of the search (as search proceeds, more information for better decision is available).

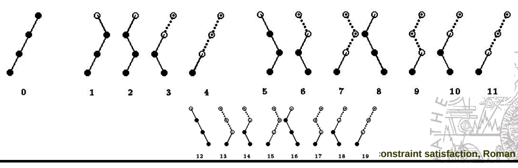
Observation 3:
The number of heuristic violations is usually small.

Limited Discrepancy Search

Discrepancy = heuristic is not followed
(a value different from the heuristic is chosen)

Idea of ***Limited Discrepancy Search*** (LDS):
- first, follow the heuristic
- when a failure occurs then explore the paths when the heuristic is not followed maximally once (start with earlier violations)
- after next failure occurs then explore the paths when the heuristic is not followed maximally twice...

Example:
the heuristic proposes to use the left branches



Algorithm LDS (Harvey, Ginsberg 1995)

```
procedure LDS-PROBE(Unlabelled,Labelled,Constraints,D)
    if Unlabelled = {} then return Labelled
    select X in Unlabelled
    Values_X ¬  D_X - {values inconsistent with Labelled using Constraints}
    if Values_X = {}  then return fail
    else select HV in Values_X using heuristic
        if D=0 then return LDS-PROBE(Unlabelled-{X}, LabelledÈ{X/HV}, Constraints, 0)
        for each value V from Values_X -{HV} do
            R ¬  LDS-PROBE(Unlabelled-{X}, LabelledÈ{X/V}, Constraints, D-1)
            if R¹ fail then return R
        end for
        return LDS-PROBE(Unlabelled-{X}, LabelledÈ{X/HV}, Constraints, D)
    end if
end LDS-PROBE

procedure LDS(Variables,Constraints)
    for D=0 to |Variables| do          % D is a number of allowed discrepancies
        R ¬  LDS-PROBE(Variables,{},Constraints,D)
        if R¹ fail then return R
    end for
    return fail
end LDS
```