Hybrid encoding

Swapping variables and constraints.

A k-ary constraint c is converted to a dual variable v, 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 a v’ restricting the dual variables to tuples in which the original shared variables take the same value.

Example:

Variables x₁,…,x₆ with domain {0,1}

C₁: x₁x₂x₃x₄ = 1
C₂: x₁x₂x₅x₆ = 1
C₃: x₄x₅x₆ = 0
C₄: x₄x₅x₆ = 0

Hidden variable encoding

New dual variables for (non-binary) constraints.

A k-ary constraint c is translated to a dual variable v, with the domain consisting of compatible tuples for each variable x in the constraint c there is a constraint between x a v, restricting tuples of dual variable to be compatible with x.

Example:

Variables x₁,…,x₆ with domains {0,1}

C₁: x₁+x₂+x₃+x₄ = 1
C₂: x₁+x₂+x₅+x₆ = 1
C₃: x₄+x₅+x₆ = 0
C₄: x₄+x₅+x₆ = 0

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 R ≠ fail 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
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=9E
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 (x).
2. Select the farthest variable from the above chosen variables for each value (y).
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 C is not satisfied by 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
Jump ← 0
for each value V from D do
    if C = consistent ((X/V)/Level) ∨ Labelled, Constraints, Level)
        Jump ← max (Jump, J)
    else
        Jump ← PreviousLevel
    end for
end if
if R = fail(Level) then return R % success or back-jump
end BJ

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

Redundant work in backtracking

What is redundant work?

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

Now, some constraint checks can be omitted:

- Y is inconsistent with X (a consistent value)
- Y is consistent with X (a previously used consistent value)

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.
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
- then a failure occurs then explore the paths when the heuristic is followed maximally once (start with earlier violations)
- after a next failure occurs then explore the paths when the heuristic is not followed maximally twice...

Example:
The heuristic proposes to use the left branches

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)
- it 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.

Algorithm LDS (Harvey, Ginsberg 1995)

procedure LDS-PROBE(Unlabelled,Labelled,Constraints,D)
if Unlabelled = {} then return Labelled
select X in Unlabelled
if R = fail then return R
if Mark(X,V) then for each value V from Values
if BT cares about the end of search (it is satisfied)
end for
end if
end for
end if

procedure consistent(X,V,Labelled,Constraints,Level)
for each Y/VY/LY in Labelled such that LY % 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
end if
end for
Mark(X,V) ← Level-1
return true
end consistent

procedure LDS(Variables,Constraints)
for D=0 to |Variables| do % D is a number of allowed discrepancies
for each Y/V in Variables do
if D=0 then return LDS-PROBE(Unlabelled-{X}, Labelled-{X/HV}, Constraints, D)
if BT cares about the end of search (it is satisfied)
end if
end for
end for
end if
end for
end consistent

procedure LDS-PROBE(Unlabelled,Labelled,Constraints,D)
if Unlabelled = {} then return Labelled
select X in Unlabelled
if R = fail then return R
if Mark(X,V) then for each value V from Values
if BT cares about the end of search (it is satisfied)
end for
end if
end for
end if

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% 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
end if
end for
Mark(X,V) ← Level-1
return true
end consistent

Algorithm backmarking

procedure BM(Unlabelled, Labelled, Constraints, Level)
if R = fail then return R
if consistent(X/V, Labelled, Constraints, Level) then
if Mark(X,V) then for each value V from D
pick first X from Unlabelled % fix order of variables
end for
end if
end if

procedure consistent(X/V, Labelled, Constraints, Level)
if BT cares about the end of search (it is satisfied)
return true
end consistent

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

Backmarking in practice

N-queens problem

1. Queens in rows are allocated to columns.
2. Latest chosen level is written next to chessboard (BackTo). At beginning 1s.
3. Farthest conflict queen at each position (MarkTo). At beginning 0s.
4. 6th queen cannot be allocated
5. Backtrack to 5, change BackTo.
6. When allocating 6th queen, all fix positions are still wrong (MarkTo=BackTo).

Example:
the heuristic proposes to use the left branches

Foundations of constraint satisfaction, Roman Barták