DMP204

SCHEDULING,
TIMETABLING AND ROUTING

Lecture 7

Constraint Programming (2)

Marco Chiarandini

Outline

Outline

A Puzzle Example

$$\begin{array}{r} \texttt{SEND} + \\ \texttt{MORE} = \\ \hline \texttt{MONEY} \end{array}$$

Two representations

- The first yields initially a weaker constraint propagation. The tree has 23 nodes and the unique solution is found after visiting 19 nodes
- The second representation has a tree with 29 nodes and the unique solution is found after visiting 23 nodes

However for the puzzle $\texttt{GERALD} + \texttt{DONALD} = \texttt{ROBERT}$ the situation is reverse. The first has 16651 nodes and 13795 visits while the second has 869 nodes and 791 visits

$\rightsquigarrow$ Finding the best model is an empirical science

Guidelines

Rules of thumbs for modelling (to take with a grain of salt):

- use representations that involve less variables and simpler constraints for which constraint propagators are readily available
- use constraint propagation techniques that require less preprocessing (ie, the introduction of auxiliary variables) since they reduce the search space better.

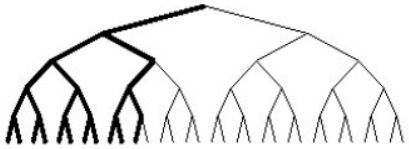
  Disjunctive constraints may lead to an inefficient representation since they can generate a large search space.
- use global constraints (see below)

Randomization in Search Tree

- Dynamical selection of solution components in construction or choice points in backtracking.
- Randomization of construction method or selection of choice points in backtracking while still maintaining the method complete
  $\rightsquigarrow$ *randomized systematic search*.
- Randomization can also be used in incomplete search

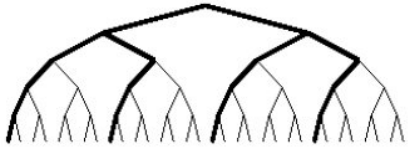
Incomplete Search

Bounded-backtrack search:



bbs(10)

Depth-bounded, then bounded-backtrack search:



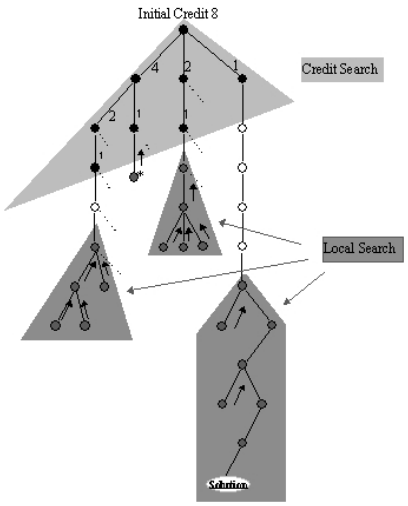
dbs(2, bbs(0))

`http://4c.ucc.ie/~hsimonis/visualization/techniques/partial_search/main.htm`

Incomplete Search

Credit-based search

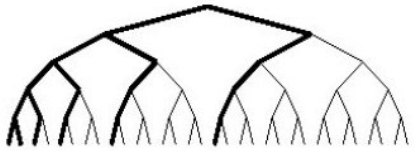
- Key idea: important decisions are at the top of the tree
- Credit = backtracking steps
- Credit distribution: one half at the best child the other divided among the other children.
- When credits run out follow deterministic best-search
- In addition: allow limited backtracking steps (eg, 5) at the bottom
- Control parameters: initial credit, distribution of credit among the children, amount of local backtracking at bottom.



Incomplete Search

Limited Discrepancy Search (LDS)

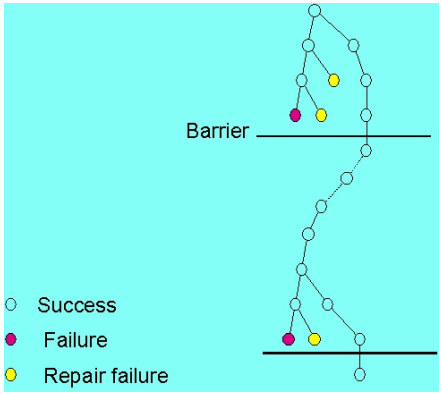
- Key observation that often the heuristic used in the search is nearly always correct with just a few exceptions.
- Explore the tree in increasing number of discrepancies, modifications from the heuristic choice.
- Eg: count one discrepancy if second best is chosen
  count two discrepancies either if third best is chosen or twice the second best is chosen
- Control parameter: the number of discrepancies



Incomplete Search

Barrier Search

- Extension of LDS
- Key idea: we may encounter several, independent problems in our heuristic choice. Each of these problems can be overcome locally with a limited amount of backtracking.
- At each barrier start LDS-based backtracking



Local Search for CSP

- Uses a complete-state formulation: a value assigned to each variable (randomly)
- Changes the value of one variable at a time
- Min-conflicts heuristic is effective particularly when given a good initial state.
- Run-time independent from problem size
- Possible use in online settings in personal assignment: repair the schedule with a minimum number of changes

Handling special constraints

Higher order constraints

Definition

Global constraints are complex constraints that are taken care of by means of a special purpose algorithm.

Modelling by means of global constraints is more efficient than relying on the general purpose constraint propagator.

Examples:

- `alldiff`
  - for m variables and n values cannot be satisfied if $m > n$,
  - consider first singleton variables
  - propagation based on bipartite matching considerations

- `cumulative` for RCPSP [Aggoun and Beldiceanu, 1993]
  - S_j starting times of jobs
  - P_j duration of job
  - R_j resource consumption
  - R limit not to be exceeded at any point in time

$$\texttt{cumulative}([S_j],[P_j],[R_j],R) := \{([s_j],[p_j],[r_j]R) \mid \forall t \sum_{i \mid s_i \leq t \leq s_i + p_i} r_i \leq R\}$$

The special purpose algorithm employes the edge-finding technique (enforce precedences)

- `sortedness` for job shop [Older, Swinkels, and van Emden, 1995]

$$\texttt{sortedness}([X_1,\ldots,X_n],[Y_1,\ldots,Yn]) := \{([d_1,\ldots,d_n],[e_1,\ldots,e_n]) \mid [e_1,\ldots,e_n] \text{ is the sorted permutation of } [d_1,\ldots,d_n]\}$$

- `among`$(x|v,l,u)$ at least l and at most v variables take values in the set v.
- `bin` $-$ `packing`$(x|w,u,k)$ pack items in k bins such that they do not exceed capacity u
- `cardinality`$(x|v,l,u)$ at least l_j and at most u_j of the variables take the value v_j
- `cardinality` $-$ `clause`$(x|k)$ $\sum_{j=1}^{n} x_j \geq k$
- `cardinality` $-$ `conditional`$(x,y|k,l)$ if $\sum_{j=1}^{n} x_j \geq k$ then $\sum_{j=1}^{m} y_j \geq l$
- `change`$(x|k,\text{rel})$ counts number of times a given change occur

- `circuit`(x) imposes Hamiltonian cycle on digraph.
- `clique`$(x|G,k)$ requires that a given graph contain a clique
- `conditional`$(\mathcal{D},\mathcal{C})$ between set of constrains $\mathcal{D} \Rightarrow \mathcal{C}$
- `cutset`$(x|G,k)$ requires that for the set of selected vertices V', the set $V \setminus V'$ induces a subgraph of G that contains no cycles.
- `cycle`$(x|y)$ select edges such that they form exactly y cycles. directed cycles in a graph.
- `diffn`$((x^1,\Delta x^1),\ldots,(x^m,\Delta x^m))$ arranges a given set of multidimensional boxes in n-space such that they do not overlap
- ...

Kinds of symmetries

- Variable symmetry:
  permuting variables keeps solutions invariant (eg, N-queens)
  $\{x_i \rightarrow v_i\} \in sol(P) \Leftrightarrow \{x_{\pi(i)} \rightarrow v_i\} \in sol(P)$
- Value symmetry:
  permuting values keeps solutions invariant (eg, GCP)
  $\{x_i \rightarrow v_i\} \in sol(P) \Leftrightarrow \{x_i \rightarrow \pi(v_i)\} \in sol(P)$
- Variable/value symmetry:
  permute both variables and values (eg, sudoku?)
  $\{x_i \rightarrow v_i\} \in sol(P) \Leftrightarrow \{x_{\pi(i)} \rightarrow \pi(v_i)\} \in sol(P)$

Symmetry

- inherent in the problem (sudoku, queens)
- artefact of the model (order of groups)

How can we avoid it?

- ... by model reformulation (eg, use set variables,
- ... by adding constraints to the model
  (ruling out symmetric solutions)
- ... during search
- ... by dominance detection

Reified constraints

- Constraints are in a big conjunction
- How about disjunctive constraints?

$$A + B = C \quad \vee \quad C = 0$$

  or soft constraints?

- Solution: reify the constraints:

$$\begin{array}{llll} (A + B = C & \Leftrightarrow & b_0) & \wedge \\ (C = 0 & \Leftrightarrow & b_1) & \wedge \\ (b_0 & \vee & b_1 & \Leftrightarrow \quad true) \end{array}$$

- These kind of constraints are dealt with in efficient way by the systems
- Then if optimization problem (soft constraints) $\Rightarrow \min \sum_i b_i$

Outline

Prolog Approach

- Prolog II till Prolog IV [Colmerauer, 1990]
- CHIP V5 [Dincbas, 1988] `http://www.cosytec.com` (commercial)
- CLP [Van Hentenryck, 1989]
- Ciao Prolog (Free, GPL)
- GNU Prolog (Free, GPL)
- SICStus Prolog
- ECL^{i}PS^e[Wallace, Novello, Schimpf, 1997] `http://eclipse-clp.org/` (Open Source)
- Mozart programming system based on Oz language (incorporates concurrent constraint programming) `http://www.mozart-oz.org/` [Smolka, 1995]

Example

The puzzle `SEND+MORE = MONEY` in ECL^{i}PS^e

```
:- lib(ic).

sendmore(Digits) :-
    Digits = [S,E,N,D,M,O,R,Y],

% Assign a finite domain with each letter - S, E, N, D, M, O, R, Y -
% in the list Digits
    Digits :: [0..9],

% Constraints
    alldifferent(Digits),
    S #\= 0,
    M #\= 0,
                  1000*S + 100*E + 10*N + D
                + 1000*M + 100*O + 10*R + E
    #= 10000*M + 1000*O + 100*N + 10*E + Y,

% Search
    labeling(Digits).
```


Other Approaches

Modelling languages similar in concept to ZIMPL:

- OPL [Van Hentenryck, 1999] ILOG CP Optimizer `www.cpoptimizer.ilog.com` (ILOG, commercial)
- MiniZinc [] (open source, works for various systems, ECL^{i}PS^e, Geocode)

MiniZinc

```
%-----------------------------------------------------------------------%
% Example from the MiniZinc paper:
% (square) job shop scheduling in MiniZinc
%-----------------------------------------------------------------------%

%-----------------------------------------------------------------------%
% Model

int: size;                                  % size of problem
array [1..size,1..size] of int: d;          % task durations
int: total = sum(i,j in 1..size) (d[i,j]);  % total duration
array [1..size,1..size] of var 0..total: s; % start times
var 0..total: end;                          % total end time

predicate no_overlap(var int:s1, int:d1, var int:s2, int:d2) =
    s1 + d1 <= s2 \/ s2 + d2 <= s1;

constraint
    forall(i in 1..size) (
        forall(j in 1..size-1) (s[i,j] + d[i,j] <= s[i,j+1]) /\
        s[i,size] + d[i,size] <= end /\
        forall(j,k in 1..size where j < k) (
            no_overlap(s[j,i], d[j,i], s[k,i], d[k,i])
        )
    );

solve minimize end;

output
    [ "jobshop_nxn\n" ] ++
    [ "s[1..",] ++ [show(size)] ++ [", 1..",] ++ [show(size)] ++ [ "] = \n  [ " ] ++
    [show(s[i,j]) ++ if j = size then if i = size then " ]\n" else "\n    " endif else " " endif | i,j in 1..size];
```

Other Approaches

Libraries:
Constraints are modelled as objects and are manipulated by means of special methods provided by the given class.

- CHOCO (free) http://choco.sourceforge.net/
- Kaolog (commercial) http://www.koalog.com/php/index.php
- ECLiPSe (free) `www.eclipse-clp.org`
- ILOG CP Optimizer `www.cpoptimizer.ilog.com` (ILOG, commercial)
- Gecode (free) `www.gecode.org` C++, Programming interfaces Java and MiniZinc
- G12 Project `http://www.nicta.com.au/research/projects/constraint_programming_platform`

CP Languages

Greater expressive power than mathematical programming

- constraints involving disjunction can be represented directly
- constraints can be encapsulated (as predicates) and used in the definition of further constrains

However, CP models can often be translated into MIP model by

- eliminating disjunctions in favor of auxiliary Boolean variables
- unfolding predicates into their definitions

CP Languages

- Fundamental difference to LP
  - language has structure (global constraints)
  - different solvers support different constraints
- In its infancy
- Key questions:
  - what level of abstraction?
    - solving approach independent: LP, CP, ...?
    - how to map to different systems?
  - Modelling is very difficult for CP
    - requires lots of knowledge and tinkering