DM204, 2010
SCHEDULING, TIMETABLING AND ROUTING

Lecture 8
Constraint Programming (2)

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Course Overview

- ✔ Problem Introduction
  - ✔ Scheduling classification
  - ✔ Scheduling complexity
  - ✔ RCPSP
- General Methods
  - ✔ Integer Programming
  - Constraint Programming
  - Heuristics
  - Dynamic Programming and Branch and Bound
- Scheduling
  - Single Machine
  - Parallel Machine and Flow Shop Models
  - Job Shop
  - Resource Constrained Project Scheduling Model
- Timetabling
  - Reservations and Education
  - University Timetabling
  - Crew Scheduling
  - Public Transports
- Vechicle Routing
  - Capacited Models
  - Time Windows models
  - Rich Models

Outline

1. Constraint Languages

2. Refinements on CP
    - Refinements: Modeling
    - Refinements: Search
    - Refinements: Constraints
    - Symmetry Breaking
    - Reification
    - CP in Scheduling

Optimization Problems

Objective function to minimize $F(X_1, X_2, \ldots, X_n)$

- Solve a modified Constraint Satisfaction Problem by setting an (upper) bound z^* in the objective function
- Dichotomic search: U upper bound, L lower bound

$$M = \frac{U + L}{2}$$

Outline

1. Constraint Languages

2. Refinements on CP
   - Refinements: Modeling
   - Refinements: Search
   - Refinements: Constraints
   - Symmetry Breaking
   - Reification
   - CP in Scheduling

Constraint Programming Systems

Expressiveness language stream
(modelling)
+
(efficient solvers)
Algorithm stream

CP systems typically include

- general purpose algorithms for constraint propagation (arc consistency on finite domains)
- built-in constraint propagation for various constraints (eg, linear, boolean, global constraints)
- built-in for constructing various forms of search

Logic Programming

Logic programming is the use of mathematical logic for computer programming.

First-order logic is used as a purely declarative representation language, and a theorem-prover or model-generator is used as the problem-solver.

Logic programming supports the notion of logical variables

- Syntax – Language
  - Alphabet
  - Well-formed Expressions
    E.g., $4X + 3Y = 10$; $2X - Y = 0$
- Semantics – Meaning
  - Interpretation
  - Logical Consequence
- Calculi – Derivation
  - Inference Rule
  - Transition System

Logic Programming

Example: Prolog

> *A logic program is a set of axioms, or rules, defining relationships between objects.*
>
> *A computation of a logic program is a deduction of consequences of the program.*
>
> *A program defines a set of consequences, which is its meaning.*

Sterling and Shapiro: The Art of Prolog, Page 1.
To deal with the other constraints one has to add other constraint solvers to the language. This led to Constraint Logic Programming

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
- ECLiPSe [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 ECLiPSe

```
:- 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

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
- 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`

Other Approaches

Modelling languages:

- OPL [Van Hentenryck, 1999] ILOG CP Optimizer
  www.cpoptimizer.ilog.com (ILOG, commercial)
- MiniZinc [] (open source, works for various systems, ECLiPSe, Geocode)
- Comet

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];
```

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

Summary

- Model your problem via Constraint Satisfaction Problem
- Declare Constraints + Program Search
- Constraint Propagation
- Languages

Outline

1. Constraint Languages

2. Refinements on CP
   - Refinements: Modeling
   - Refinements: Search
   - Refinements: Constraints
   - Symmetry Breaking
   - Reification
   - CP in Scheduling

Modelling

- Different views to the problem
- Adding implied constraints
- Auxiliary variables to make it easier to state constraints and improve constraint propagation

A Puzzle Example

$$\begin{array}{r} \texttt{SEND}\ + \\ \texttt{MORE}\ = \\ \texttt{MONEY} \end{array} \qquad\qquad \begin{array}{r} \texttt{GERALD}\ + \\ \texttt{DONALD}\ = \\ \texttt{ROBERT} \end{array}$$

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

$\leadsto$ 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)

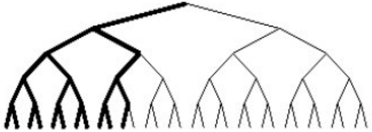
- Backtracking
- Branch and Bound
- Local Search

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

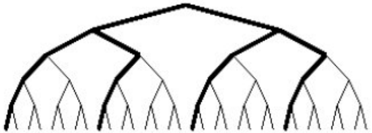
Incomplete Search

Bounded-backtrack search:



bbs(10)

Depth-bounded, then bounded-backtrack search:



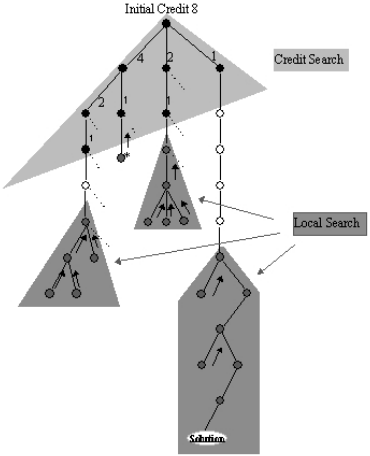
dbs(2, bbs(0))

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

Incomplete Search

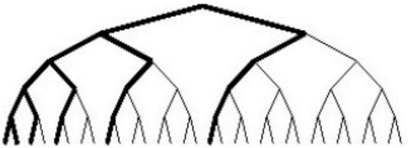
Credit-based search

- 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)

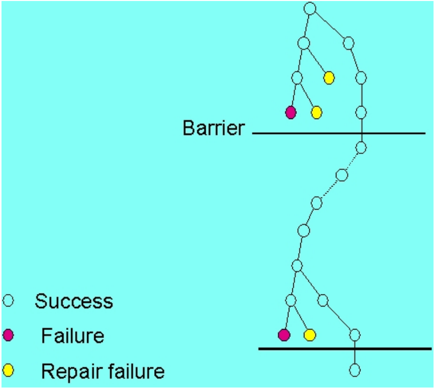


- Key observation that often the heuristic used in the search is nearly always correct with just a few exceptions.
- Explore the tree in increasing number of discrepancies, modifications from the heuristic choice.
- Eg: count one discrepancy if second best is chosen
  count two discrepancies either if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



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

- `sum`(x, z, c): $z = \sum_i c_i x_i$
- `knapsack`(x, z, c): $\min D(z) \leq \sum_i c_i x_i \leq \max D(z)$
- `binpacking`$(x|w, u, k)$ pack items in k bins such that they do not exceed capacity u
- `alldifferent`$(x) = \{(d_1, \ldots, d_n) | \forall_i d_i \in D(x_i), \forall_{i \neq j} d_i \neq d_j\}$
- `element`$(y, z, x) : \{e, f, d_1, \ldots, d_n) | e \in D(y), f \in D(z), \forall_i d_i \in D(x_i), f = d_e\}$
  aka: `channeling`
- `change`$(x|k, \mathrm{rel})$ k be the number of times two consecutive variables x_i, x_{i+1} satisfy $x_i \mathrm{\ rel\ } x_{i+1}$

- $\texttt{gcc}(x_1, \ldots, x_n, c_{v_1}, \ldots, c_{v_m}) =$ the number of occurrences of v_j in $d \in D(X)$ is in $D(c_{v_j})$

  aka:

  $\texttt{cardinality}(l, x, u)$ if there are at least l_i variables in array x that are assigned value v_i and at most up_j variables in array x that are assigned value v_j.

  $\texttt{cardinality}(x|v, l, u)$ at least l_j and at most u_j of the variables take the value v_j

  $\texttt{among}(x|v, l, u)$ at least l and at most v variables take values in the set v.

  atmost, atleast among

- `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
- ...

- `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)

- `atmost` Resource Constraint
  - check the sum of minimum values of single domains
    delete maximum values if not consistent with minimum values of others.
  - for large integer values not possible to represent the domain as a set of integers but rather as bounds.
    Then bounds propagation: Eg,

    $\texttt{Flight271} \in [0, 165]$ and $\texttt{Flight272} \in [0, 385]$
    $\texttt{Flight271} + \texttt{Flight272} \in [420, 420]$
    $\texttt{Flight271} \in [35, 165]$ and $\texttt{Flight272} \in [255, 385]$

Global Constraints Catalogue

http://www.emn.fr/x-info/sdemasse/gccat/

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:

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

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

Scheduling Models

- Variable for start-time of task a start(a)
- Precedence constraint:
  $start(a) + dur(a) \leq start(b)$ (a before b)
- Disjunctive constraint:
  $start(a) + dur(a) \leq start(b)$ (a before b)
  or
  $start(b) + dur(b) \leq start(a)$ (b before a)
  Solved by reification
- Cumulative Constraints (renewable resources)
  For tasks a and b on resource R
  $use(a) + use(b) \leq cap(R)$
  or $start(a) + dur(a) \leq start(b)$
  or $start(b) + dur(b) \leq start(a)$

Propagators for Scheduling

Serialization: ordering of tasks on one machine

- Consider all tasks on one resource
- Deduce their order as much as possible
- Propagators:
  - Timetabling: look at free/used time slots
  - Edge-finding: which task first/last?
  - Not-first / not-last

Job Shop Problem

- Hard problem!
- 6x6 instance solvable using Gecode
  - disjunction by reification
  - normal branching
- Classic 10x10 instance not solvable using Gecode!
  - specialized propagators (edge-finding) and branchings needed