DM204, 2010
SCHEDULING, TIMETABLING AND ROUTING

Lecture 7
Constraint Programming

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

Outline

1. Constraint Programming
   - Constraint Satisfaction Problem
   - General Purpose Solvers
2. Constraint Languages
3. Refinements on CP
   - Refinements: Modeling
   - Refinements: Search

Constraint Programming

Constraint Programming is about a fomrulation of the problem as a constraint satisfaction problem and about solving it by means of general or domain specific methods.

constraint declaration
declarative language
(MODEL)
+
(SEARCH)
procedural language
general purpose constraint solver

Overview

Handbook of Constraint Programming. F. Rossi, P. van Beek and T. Walsh (ed.). Handbook of Constraint Programming, Elsevier, 2006

- Constraint Propagation
- Search
- Global Constraints
- Complexity and Tractable Cases
- Soft Constraints
- Symmetries
- Modelling
- Integration with OR
- Continuous and Structured Domains
- Dynamic and Uncertain Environments

Constraint Satisfaction Problem

- **Input:**
  - a set of variables $X_1, X_2, \ldots, X_n$
  - each variable has a non-empty domain D_i of possible values
  - a set of constraints. Each constraint C_i involves some subset of the variables and specifies the allowed combination of values for that subset.

    [A constraint C on variables X_i and X_j, $C(X_i, X_j)$, defines the subset of the Cartesian product of variable domains $D_i \times D_j$ of the consistent assignments of values to variables. A constraint C on variables X_i, X_j is satisfied by a pair of values v_i, v_j if $(v_i, v_j) \in C(X_i, X_j)$.]
- **Task:**
  - find an assignment of values to all the variables $\{X_i = v_i, X_j = v_j, \ldots\}$
  - such that it is consistent, that is, it does not violate any constraint

If assignments are not all equally good, but some are preferable this is reflected in an objective function.

Solution Process

Standard search problem:

- initial state: the empty assignment $\{\}$ in which all variables are unassigned
- successor function: a value can be assigned to any unassigned variable, provided that it does not conflict with previous assignments
- goal test: the current assignment is complete

Two fundamental issues:

- exploration of search tree (of depth n)
- constraint propagation (via domain filtering algorithm)
  - at every node of the search tree, remove domain values that do not belong to a solution
  - repeat until nothing can be removed anymore

$\leadsto$ In CP, we mostly mean complete search but incomplete search is also included.

Constraint Propagation

Definition (Domain consistency)

A constraint C on the variables $X_1, \ldots, X_k$ is called domain consistent if for each variable X_i and each value $v_i \in D(X_i)$ $(i = 1, \ldots, k)$, there exist a value $v_j \in D(X_j)$ for all $j \neq i$ such that $(d_1, \ldots, d_k) \in C$.

- domain consistency = hyper-arc consistency or generalized-arc consistency
- Establishing domain consistency for unary and binary constraints is inexpensive.
- For higher arity constraints the naive approach requires time that is exponential in the number of variables.
- Exploiting underlying structure of a constraint can sometimes lead to establish domain consistency much more efficiently.

Constraint Propagation

Alternative names:

- constraint relaxation
- filtering algorithms
- narrowing algorithms
- constraint inference
- simplification inference
- label inference
- local consistency enforcing
- rule iteration

Types of Variables and Values

- Discrete variables with finite domain:
  complete enumeration is $O(d^n)$
- Discrete variables with infinite domains:
  Impossible by complete enumeration.
  Instead a constraint language (constraint logic programming and constraint reasoning)
  Eg, project planning.

  $$S_j + p_j \leq S_k$$

  NB: if only linear constraints, then integer linear programming
- Variables with continuous domains
  branch and reduce
  NB: if only linear constraints or convex functions then mathematical programming
- structured domains (eg, sets, paths, multisets)

Constraint Reasoning

Combination

Simplification

Contradiction

Redundancy

Types of Constraints

- Unary constraints $C(X_i)$
- Binary constraints (constraint graph) $C(X_i, X_j)$
- Higher order (constraint hypergraph) $C(X_i, \ldots, X_k)$
  (k arbitrary number)

  Global constraints or combinatorial cosntraints
  ease the task of modelling
  admit efficient specialized algorithms

  Eg, `alldifferent()`, `among()`, etc.
- Soft constraints
- Preference constraints
  cost on individual variable assignments

General Purpose Algorithms

Search algorithms

organize and explore the search tree

- Search tree with branching factor at the top level nd and at the next level $(n-1)d$. The tree has $n! \cdot d^n$ leves even if only d^n possible complete assignments.
- Insight: CSP is commutative in the order of application of any given set of action (the order of the assignment does not influence)
- Hence we can consider search algs that generate successors by considering possible assignments for only a single variable at each node in the search tree.
  The tree has d^n leaves.

Backtracking search

depth first search that chooses one variable at a time and backtracks when a variable has no legal values left to assign.

Backtrack Search

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
   return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment according to CONSTRAINTS[csp] then
         add {var = value} to assignment
         result ← RECURSIVE-BACKTRACKING(assignment, csp)
         if result ≠ failure then return result
         remove {var = value} from assignment
   return failure
```

Backtrack Search

- No need to copy solutions all the times but rather extensions and undo extensions
- Since CSP is standard then the alg is also standard and can use general purpose algorithms for initial state, successor function and goal test.
- Backtracking is uninformed and complete. Other search algorithms may use information in form of heuristics

General Purpose Backtracking

Implemnetation Refinements

1) [Search] Which variable should we assign next, and in what order should its values be tried?

2) [Propagation] What are the implications of the current variable assignments for the other unassigned variables?

3) [Search] When a path fails – that is, a state is reached in which a variable has no legal values can the search avoid repeating this failure in subsequent paths?

Search

1) Which variable should we assign next, and in what order should its values be tried?

- Select-Initial-Unassigned-Variable
  degree heuristic (reduces the branching factor) also used as tied breaker
- Select-Unassigned-Variable
  Most constrained variable (DSATUR) = fail-first heuristic
  = Minimum remaining values (MRV) heuristic (speeds up pruning)
- Order-Domain-Values
  least-constraining-value heuristic (leaves maximum flexibility for subsequent variable assignments)

NB: If we search for all the solutions or a solution does not exists, then the ordering does not matter.

Propagation

2) What are the implications of the current variable assignments for the other unassigned variables?

Propagating information through constraints

- Implicit in Select-Unassigned-Variable
- Forward checking (coupled with MRV)
- Constraint propagation (filtering)

Constraint Propagation

- node consistency
- arc consistency: force all (directed) arcs uv to be consistent:
  $\exists$ a value in $D(v)$: $\forall$ values in $D(u)$, otherwise detects inconsistency

  can be applied as preprocessing or as propagation step after each assignment (MAC, Maintaining Arc Consistency)

  applied repeatedly

- k-consistency: if for any set of $k-1$ variables, and for any consistent assignment to those variables, a consistent value can always be assigned to any k-th variable.

determining the appropriate level of consistency checking is mostly an empirical science.

Example: Arc Consistency Algorithm AC-3

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains
  inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**
      for each X_k **in** NEIGHBORS[X_i] **do**
        add (X_k, X_i) to *queue*

---

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff we remove a value
  removed $\leftarrow$ *false*
  for each x **in** DOMAIN[X_i] **do**
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint between X_i and X_j
      then delete x from DOMAIN[X_i]; *removed* $\leftarrow$ *true*
  return *removed*

Search

3) When a path fails – that is, a state is reached in which a variable has no legal values can the search avoid repeating this failure in subsequent paths?

Backtracking-Search

- chronological backtracking, the most recent decision point is revisited
- backjumping, backtracks to the most recent variable in the conflict set (set of previously assigned variables connected to X by constraints).

  every branch pruned by backjumping is also pruned by forward checking

  idea remains: backtrack to reasons of failure.

An Empirical Comparison

Median number of consistency checks

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV
USA	(> 1,000K)	(> 1,000K)	2K	60
n-Queens	(> 40,000K)	13,500K	(> 40,000K)	817K
Zebra	3,859K	1K	35K	0.5K
Random 1	415K	3K	26K	2K
Random 2	942K	27K	77K	15K

The structure of problems

- Decomposition in subproblems:
  - connected components in the constraint graph
  - $O(d^c n/c)$ vs $O(d^n)$
- Constraint graphs that are tree are solvable in poly time by reverse arc-consistency checks.
- Reduce constraint graph to tree:
  - removing nodes (cutset conditioning: find the smallest cycle cutset. It is NP-hard but good approximations exist)
  - collapsing nodes (tree decomposition)
    divide-and-conquer works well with small subproblems

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 Programming
   - Constraint Satisfaction Problem
   - General Purpose Solvers
2. Constraint Languages
3. Refinements on CP
   - Refinements: Modeling
   - Refinements: Search

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
- Decalre Constraints + Program Search
- Constraint Propagation
- Languages

Outline

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

```
 SEND +
 MORE =
MONEY
```

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 GERALD + DONALD = ROBERT the situation is reverse. The first has 16651 nodes and 13795 visits while the second has 869 nodes and 791 visits

⇝ 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)

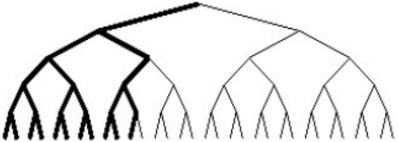
- Backtracking
- Branch and Bound
- Local Search

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

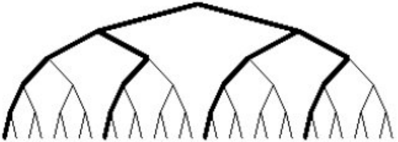
Incomplete Search

Bounded-backtrack search:



bbs(10)

Depth-bounded, then bounded-backtrack search:



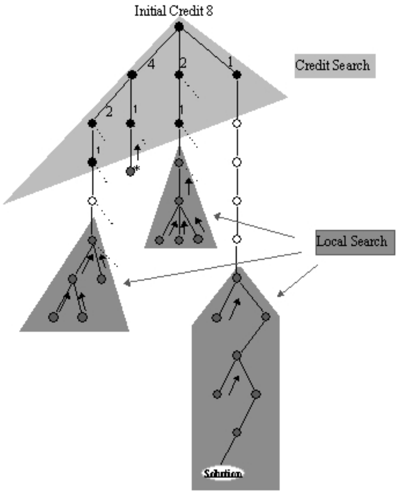
dbs(2, bbs(0))

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

Incomplete Search

Credit-based search

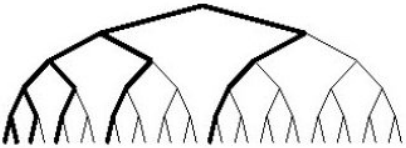
- Key idea: important decisions are at the top of the tree
- Credit = backtracking steps
- Credit distribution: one half at the best child the other divided among the other children.
- When credits run out follow deterministic best-search
- In addition: allow limited backtracking steps (eg, 5) at the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



Incomplete Search

Limited Discrepancy Search (LDS)

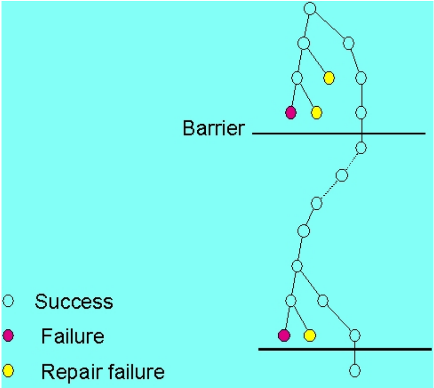
- Key observation that often the heuristic used in the search is nearly always correct with just a few exceptions.
- Explore the tree in increasing number of discrepancies, modifications from the heuristic choice.
- Eg: count one discrepancy if second best is chosen
  count two discrepancies either if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



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