DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 6
MIP Modelling and Constraint Programming

Marco Chiarandini

Outline

Outline

Position variables

$Qm \mid p_j = 1 \mid \sum h_j(C_j)$, h_j non decreasing function

model as a transportation problem

$$x_{ijk} \geq 0 \qquad \forall i = 1, \dots, m,\ j, k = 1, \dots, n$$

Variables indicate if j is scheduled as the kth job on the machine i.
No need to declare them binary

$$\sum_{i=1}^{m}\sum_{k=1}^{n} x_{ijk} = 1 \qquad \forall j = 1, \dots, n$$

Every job assigned to one only position

$$\sum_{j=1}^{n} x_{ijk} \leq 1 \qquad \forall i = 1, \dots, m, k = 1, \dots, n$$

At most one job can be processed in time

$$\min \sum_{j=1}^{n}\sum_{i=1}^{m}\sum_{k=1}^{n} c_{ijk} x_{ijk}$$

Objective, $c_{ijk} = h_j(C_j) = h_j(k/v_i)$

Time indexed variables

$1|r_j|\sum w_j C_j$
Discretize time in $t = 0, \ldots, l$, where l is upper bound

$x_{jt} \in \{0,1\} \qquad j = 1, \ldots, n;\ t = 0, \ldots, l$ — Variables indicate if j starts at t

$\sum_{t=1}^{l} x_{jt} = 1 \qquad \forall j = 1, \ldots, n$ — Every job starts at one point in time

$\sum_{j=1}^{n} \sum_{s=\max\{t-p_j,0\}}^{t-1} x_{js} \leq 1 \qquad \forall t = 0, \ldots, l$ — At most one job can be processed in time

$x_{jt} = 0 \quad \forall j = 1, \ldots, n,\ t = 0, \ldots, \max\{r_j - 1, 0\}$ — Jobs cannot start before their release dates

$\min \sum_{j=1}^{n} \sum_{t=0}^{l} w_j (t + p_j) x_{js}$ — Objective

Sequencing variables

$1|prec|\sum w_j C_j$

$x_{jk} \in \{0,1\} \qquad j, k = 1, \ldots, n$ — Variables indicate if j precedes k

$x_{jj} = 0 \qquad \forall j = 1, \ldots, n$

$x_{kj} + x_{jk} = 1 \qquad \forall j, k = 1, \ldots, n, j \neq k$ — Precedence constraints

$x_{kj} + x_{lk} + x_{jl} \geq 1 \qquad j, k, l = 1, \ldots, n, j \neq k, k \neq l, j \neq l$ — Precedence constraints

$\min \sum_{j=1}^{n} \sum_{k=1}^{n} w_j p_k x_{kj} + \sum_{j=1}^{n} w_j p_j$ — Objective

Real Variables

Disjunctive Programming

$1|prec|\sum w_j C_j$

Disjunctive graph model made of conjunctive arcs A and disjunctive arcs I.
Select disjunctive arcs such that the graph does not contain a cycle.

$x_j \in \mathbf{R} \qquad j = 1, \ldots, n$ — Variables denote completion of job j

$x_k - x_j \geq p_k \qquad \forall j \rightarrow k \in A$ — precedence constraints conjunctive arcs

$x_j \geq p_j \qquad \forall j = 1, \ldots, n$ — min processing time

$x_k - x_j \geq p_k$ or $x_j - x_k \geq p_j \qquad \forall (i,j) \in I$ — disjunctive constraints

$\min \sum_{j=1}^{n} w_j x_j$ — Objective

Linearizations

How to linearize these non linear functions?

- Disjunctive constraints
- $\min |a - b|$
- $\min\{\max(a, b)\}$
- $\min \max_{i=1,\ldots,m}(c_i^T x + d_i)$ piecewise-linear functions

Constraint types

Constraint type	Normalized representation
Set partitioning	$\sum x_i = 1$
Set packing	$\sum x_i \leqslant 1$
Set covering	$\sum x_i \geqslant 1$
Cardinality constraint	$\sum x_i = b$
Bin packing	$\sum a_i x_i + a_k x_k \leqslant a_k$
Invariant knapsack	$\sum x_i \leqslant b$
Knapsack	$\sum a_i x_i \leqslant b$
Integer knapsack	$\sum a_i y_i \leqslant b$
Variable lower bound	$p_k x_k - z_k \leqslant 0$ (or $p_k x_k - y_k \leqslant 0$)
Variable upper bound	$p_k x_k - z_k \geqslant 0$ (or $p_k x_k - y_k \geqslant 0$)
Mixed binary constraint	$\sum p_i x_i + \sum r_i z_i \leqslant t$ (or $=t$)
General constraint	$\sum p_i x_i + \sum q_i y_i + \sum r_i z_i \leqslant t$ (or $=t$)

x binary, y general integer, z a continuous variable.
a and b integer numbers; p, q, r, s real numbers

- Specific domain propagation, preprocessing and cut generation exist for some of these constraints.
  [Achterberg, T. Constraint Integer Programming Department of Mathematics, Phd Thesis, Technical University of Berlin, Germany, 2007]

Outline

Constraint Programming

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

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
- path cost: a constant cost for every step.

Two fundamental issues:

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

$\rightsquigarrow$ The search may be both complete and incomplete.

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

  NB: if only linear constraints or convex functions then mathematical programming

Constraint Propagation

Definition

A constraint C on the variables $x_1, \ldots, x_k$ is called domain consistent if for each variable x_i and each value $d_i \in D(x_i)$ $(i = 1, \ldots, k)$, there exist a value $d_j \in D(x_j)$ 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 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.

Types of constraints

- Unary constraints
- Binary constraints (constraint graph)
- Higher order (constraint hypergraph)
  Eg, `alldiff()`, `among()`, etc.

  Every higher order constraint can be reconduced to binary (you may need auxiliary 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 $n! \cdot d^n$ leaves 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

Implementation Refinements

1) Which variable should we assign next, and in what order should its values be tried?
2) What are the implications of the current variable assignments for the other unassigned variables?
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?

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.

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)
  - 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 {X1, X2, ..., Xn}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (Xi, Xj) <- REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
      for each Xk in NEIGHBORS[Xi] do
        add (Xk, Xi) to queue

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


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 $F(X_1, X_2, \dots, X_n)$

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

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

- Reified constraints (more later)

CP Systems

Programming language + Systems

The system typically includes

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

Constraints are added to a **constrain store** to which various constraint solvers are attached.
$\leadsto$ Constraint variables are unknowns in mathematical sense.

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.

- 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

$\rightsquigarrow$ Logic programming supports the notion of logical variables

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

A Puzzle Example

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

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 $\mathtt{GERALD} + \mathtt{DONALD} = \mathtt{ROBERT}$ 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

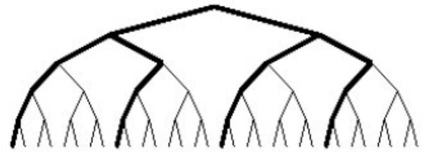
- Dynamical selection of solution components in construction or choice points in backtracking.
- Randomization of construction method or selection of choice points in backtracking while still maintaining the method complete
  $\rightsquigarrow$ *randomized systematic search*.
- Randomization can also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

Incomplete Search

Bounded-backtrack search:

bbs(10)

Depth-bounded, then bounded-backtrack search:



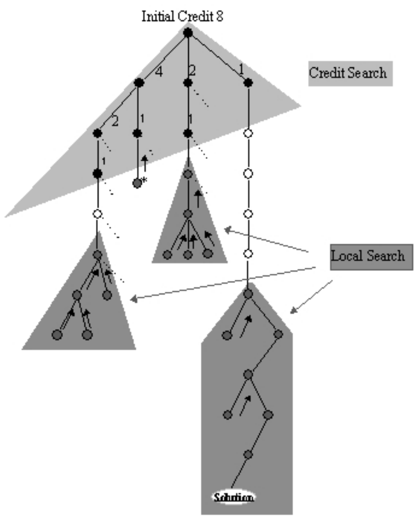
dbs(2, bbs(0))

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

Incomplete Search

Credit-based search

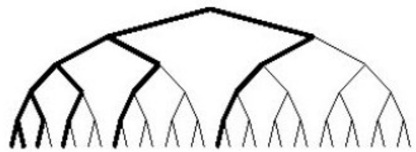
- Key idea: important decisions are at the top of the tree
- Credit = backtracking steps
- Credit distribution: one half at the best child the other divided among the other children.
- When credits run out follow deterministic best-search
- In addition: allow limited backtracking steps (eg, 5) at the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



Incomplete Search

Limited Discrepancy Search (LDS)

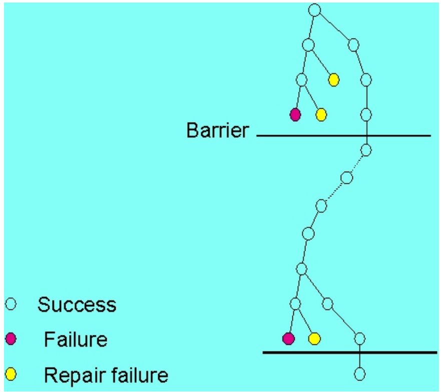
- Key observation that often the heuristic used in the search is nearly always correct with just a few exceptions.
- Explore the tree in increasing number of discrepancies, modifications from the heuristic choice.
- Eg: count one discrepancy if second best is chosen
  count two discrepancies either if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