Advanced Techniques in Constraint Programming

Krzysztof Kuchcinski
Dept. of Computer Science, LTH
Outline

Soft Constraints

Symmetry Elimination
  Problem reformulation
  Symmetry breaking constraints
  Adaptation of search algorithm
  Heuristic methods for building search trees

Connection to Integer Linear Programming

Local Search

Simulated Annealing

Tabu search
Soft Constraints
Soft Constraints

- Hard and soft constraints
- Hard constraints *must* be satisfied in all solutions.
- Soft constrained, on the other hand, *may be violated* in admissible solution.
- can be handled using a test of entailment (e.g., reified constraint) and optimization of satisfaction criteria for soft constraints.
- disadvantage of this solution is that constraints are passively watched whether they are satisfied or not; no direct pruning.
Soft Constraints (cont’d)

Example

- Consider task assignment problem that assigns tasks to processors while minimizing the execution time for the whole set of tasks.
- Assume that we would like to have the optimal execution time of 3 but we would like to assign task pairs (1, 2), (7, 9), and (2, 5) to the same processor.

\[
\text{impose } (r_1 \neq r_2) \iff b_1 \\
\text{impose } (r_7 \neq r_9) \iff b_2 \\
\text{impose } (r_2 \neq r_5) \iff b_3
\]

- Solution with the maximal number of satisfied constraints is found by minimizing cost function \( b_1 + b_2 + b_3 \)
- solution— \( b_1 = 0, b_2 = 0, \) and \( b_3 = 1 \)
Soft Constraints (cont’d)

**Example**

- $C$ is the soft constraint $x \leq y$ and $cost$ quantifies it violation
  
  \[
  cost = \begin{cases} 
  0, & \text{if } C \text{ is satisfied} \\ 
  x - y, & \text{if } C \text{ is violated} 
  \end{cases}
  \]

- $D_x :: \{90001..100000\}$, $D_y :: \{0..200000\}$, and $cost \leq 5$ that implies $x - y \leq 5$,

- Result– $D_y :: \{89996..200000\}$

- soft constrained definition for the example:
  
  \[
  (x \leq y \land cost = 0) \lor (cost = x - y \land cost > 0)
  \]
Symmetry Elimination
Symmetry Elimination

Different symmetrical assignments of tasks to three processors example.
Symmetries

- Symmetries can be either inherited in the problem or introduced into the problem because of the modeling style.
- The size of the search tree becomes very large.
- It can be difficult to prove optimality.
- Avoid modeling styles that introduces symmetries or find the ways to avoid repeatedly visit symmetrical solutions during search.
**Definition**

**Definition (Symmetry, Puget 2002)**

A symmetry $\sigma$ for CSP $S = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ is a one to one mapping (bijection) from decisions to decisions of $S$ s.t.

i) for all assignments, $A = \{v_i = d_i\}$, $\sigma(a) = \{\sigma(v_i = d_i)\}$

ii) for all assignments $A$, $a \in \text{sol}(S)$ iff $\sigma(A) \in \text{sol}(S)$

where $\text{sol}(S)$ denotes the set of all solutions of $S$.

- Symmetrical solution can be obtained by the following bijection
  $\sigma(r_i = 1) = (r_i = 1), \sigma(r_i = 2) = (r_i = 3), \sigma(r_i = 3) = (r_i = 2)$, i.e. swap tasks between processors 2 and 3.
- two types of symmetry
  - value symmetry and
  - variable symmetries.
Symmetry Breaking

Four main approaches to symmetry breaking:

- reformulate the problem,
- add symmetry breaking constraints before search,
- adapt search algorithm to break symmetry, and
- build search tree so that no symmetry arises.
Problem reformulation

- Problem dependent,
- Dependent on available constraints,
- Not always possible.
Symmetry breaking constraints

- Adds special constraints that prevent the solver from exploring symmetrical solutions.
- One of the first approached proposed to eliminate symmetries.
- Combinatorial constraints based on lexicographical order have been proposed.
- Lexicographical order defines a total order on solutions and therefore breaks symmetries.
Symmetry breaking constraints (cont’d)

Example

- Consider all 6 symmetrical solutions as presented before.
- The solution can be represented as a matrix over tasks and processors where 1 on position $i, j$ means that task $j$ is assigned to processor $i$.

<table>
<thead>
<tr>
<th>Task</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$p_2$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$p_3$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Symmetry breaking constraints (cont’d)

Example (cont’d)

- generate values $b_{i,j}$ using reified constraints of the form $r_i = j \Leftrightarrow b_{i,j}$ and then impose constraints.

\[
\begin{align*}
b_{1,10} \cdot 2^0 + b_{1,9} \cdot 2^1 + \cdots + b_{1,1} \cdot 2^9 &= v_1 \\
b_{2,10} \cdot 2^0 + b_{2,9} \cdot 2^1 + \cdots + b_{2,1} \cdot 2^9 &= v_2 \\
b_{3,10} \cdot 2^0 + b_{3,9} \cdot 2^1 + \cdots + b_{3,1} \cdot 2^9 &= v_3 \\
v_1 &\leq v_2, v_2 \leq v_3
\end{align*}
\]

- these constraints reduces the number of solutions by factor of 6, from 72 to 12.
Lexicographical order constraint

Example

```
lex_less([b[1,j] | i in 1..10], [b[2,j] | i in 1..10])
lex_less([b[2,j] | i in 1..10], [b[3,j] | i in 1..10])
```

or

```
lex2(b)
```

JaCoP constraint LexOrder.
Adaptation of search algorithm

- The method assumes that symmetries existing in our problem can be formally defined.
- A search algorithm will not visit parts of the search tree that perform symmetrical assignments (the symmetrical sub-tree do not need to be visited again).
- There are basically two methods to implement this approach
  - constraints are added during search; they rule out visiting equivalent nodes in future.
  - dominance detection (every node is checked before entering it and it is not entered if an equivalent node has been visited before).

Note

Both methods implement in some sense computational group theory (CGT) methods to detect dominance.
Example (Graph coloring problem)

- Limit the number of values that has to be explored by the search procedure by trying the “representative” values that do not rule out the solution but break symmetries.
- For the graph coloring problem, the rule simply defines how to color vertex $v$
  - color $v$ with one of colors that are already used, or
  - color $v$ with an arbitrary color that has not yet been used.
- This method breaks all $n!$ symmetries.
Connection to Integer Linear Programming
Linear Programming

Definition (Standard Form of Linear Programming)

The linear programming standard form is defined as

\[
\min \; z = \sum_{j=1}^{n} c_j x_j
\]

subject to

\[
\sum_{j=1}^{n} a_{ij} x_j = b_i \quad i = 1..m
\]

\[
x_j \geq 0 \quad j = 1..n
\]

where \(x_i\) are variables to be solved and \(a_{ij}, c_j\) and \(b_i\) are known coefficients.
Linear Programming (cont’d)

- LP tools allow to define the problem using inequalities and variables that are not limited to non-negative values.
- The problem is rewritten using the following rules:
  - each $x_i$ variable that is not non-negative is replaced by expression $x_i^+ - x_i^-$, where $x_i^+$ and $x_i^-$ are two new non-negative variables,
  - each inequality of the form $e \leq r$ where $e$ is a linear expression and $r$ is a number can be replaced with $e + s = r$ where $s$ is a new non-negative slack variable.
- Linear programming problems are solvable in *polynomial time*.
- Many solvers use, however, the *simplex algorithm* that has an exponential complexity in the worst case (such cases seem never to be encountered in practical applications).
Linear Programming Example

\[
\begin{align*}
\text{max } & 12 \cdot x + 20 \cdot y \\
\text{subject to} & \\
0.2 \cdot x + 0.4 \cdot y & \leq 400 \\
0.5 \cdot x + 0.4 \cdot y & \leq 490 \\
x & \geq 100 \\
y & \geq 100
\end{align*}
\]

Optimal solution, cost=20600
Integer Linear Programming

- Integer linear programming (IP) is a subclass of the linear programming where the decision variables are limited to take integer values.
- If binary decision variables are used instead of integer ones, the problem is called 0/1 linear programming problem.
- The optimal LP solution is in general fractional: violates the integrality constraint
- A simple method for solving IP problem can be constructed using simplex algorithm together with branch-and-bound search.
Simplex algorithm
Branch and Bound based LP

Example

Consider a simple IP formulation below.

\[
\begin{align*}
\text{min} & \quad 7 \cdot x_1 + 12 \cdot x_2 + 5 \cdot x_3 + 14 \cdot x_4 \\
\text{subject to} & \quad 300 \cdot x_1 + 600 \cdot x_2 + 500 \cdot x_3 + 1600 \cdot x_4 \geq 700 \\
& \quad x_1 \leq 1 \\
& \quad x_2 \leq 1 \\
& \quad x_3 \leq 1 \\
& \quad x_4 \leq 1 \\
& \quad x_1, x_2, x_3, x_4 \text{ non negative integer}
\end{align*}
\]
Branch and Bound based LP

Example (cont’d)

sol = (0,0,0,0.44), cost=6.12

\[ x_4 \leq 0 \] \quad \rightarrow \quad \text{sol = (0,0.33,1,0), cost=9}
\[ x_4 \geq 1 \]

\[ x_2 \leq 0 \] \quad \rightarrow \quad \text{infeasible}
\[ x_2 \geq 1 \]

\[ x_1 \leq 0 \] \quad \rightarrow \quad \text{infeasible}
\[ x_1 \geq 1 \]

\[ x_3 \leq 0 \] \quad \rightarrow \quad \text{sol = (1,0,0,8,0), cost=11}
\[ x_3 \geq 1 \]

\[ x_3 \geq 1 \]

sol = (0,0,1,0), cost=14

sol = (1,0,1,0), cost=12
Cutting Planes Algorithm

- Iterative procedure:
  - solving a linear relaxation of the problem $P$, $x^*$ optimal solution
  - add cutting planes when the optimal solution of the relaxation is not integral

- Cutting Planes:
  - linear inequalities $\alpha x \leq \alpha_0$
    - should cut off the optimal solution of the Linear Relaxation
      » $\alpha x^* > \alpha_0$
    - should not remove any integer solution $\rightarrow$ valid cut
      » $\alpha x \leq \alpha_0 \ \forall x \in \text{conv}(P)$ where $\text{conv}(P)$ is the convex hull of $P$
Cutting Planes Algorithm

- Cutting Planes: syntactic cuts, do not exploit the problem structure
- Convergence is not guaranteed in general
- For some cases, i.e., Gomory cuts, the process converges but it can be too expensive
Polyhydral Cuts

- Problem structure dependent
- Given an Integer Problem: $S$ is the set of its solutions
  - $\text{conv}(S)$: convex hull of $S$
  - if we have a constraint representation of the convex hull we can optimally solve the IP with Linear Programming
  - impossible to find the $\text{conv}(S)$ efficiently
- Idea: generate cuts that are facets of the convex hull
Branch and Cut

- Integrates Branch & Bound and Cutting Planes
- Two step tree search procedure: at each node
  - solving a relaxation of the original problem
  - add cuts when the optimal solution of the relaxation is not integral in order to improve the bound
- Branch when cuts are no longer effective
- In Branch & Cut in general we have a unique pool of cuts globally valid
Comparison CP and MILP

- Optimization:
  - In CP each time a feasible solution is found $Z^*$, a constraint is added on the objective function variable $Z < Z^*$. Since $Z$ is linked to problem variables, propagation is performed. At each node, when variables are instantiated and propagation is performed, bounds of $Z$ are updated.
  - In MIP, at each node the LP relaxation is solved providing a lower bound on the problem. If the lower bound is worse than the current upper bound, the node is fathomed. Otherwise, a non integral variable $x$ is selected and the branching is performed on its bounds. An initial upper bound can be in general computed.
The main motivations for integrating modeling and solving techniques from CP and MIP are:

- Combine the advantages of the two approaches
  - CP: modeling capabilities, interaction among constraints
  - MIP: global reasoning on optimality, solution methods

- Overcome the limitations of both
  - CP: poor reasoning on the objective function
  - MIP: not flexible models, no symbolic constraints
Integration

- two generic cooperative schemes: decomposition scheme and multiple search scheme.
  - decomposition scheme
    - the original problem is decomposed into a number of sub-problems and applies a “slave” solver to them.
    - the partial results are collected by a master and used to reduce the search space and guide the search for the original problem.
    - Most often the “slave” solvers provide a new tightened bounds for variables.
  - multiple search scheme
    - two or more solvers are used to solve the original problem.
    - the solvers are run in parallel and communicate their results (e.g., IP and CP).
Decomposition

Example (Use of LP solver in CP)

\[
C = \{ \\
-3x_1 + 2x_2 \leq 0, \\
3x_1 + 2x_2 \leq 6, \\
x_1 \leq 2, x_2 \leq 2, \\
x_1 \geq 0, x_2 \geq 0, \\
\text{integral}([x_1, x_2]) \\
\}
\]

- traditional bounds consistency \(x_1 :: 0..2, x_2 :: 0..2\).
- application of simplex algorithm can narrow domains further.
- for example, the upper bound of \(x_2\) is 1.5 and thus the maximal value in the domain of \(x_2\) can be reduced to 1.
Local Search
Local search

- CP search is based on depth-first-search (DFS) principle.
- CP search can stack at a part of the tree for large problems.
- There exist a set of local search methods that are based on different search principles than DFS.
- The local search starts with an initial solution and then it tries to improve it by applying local transformations.
Local search

// Step 1 - Initialization
Select a starting solution $x_{\text{current}} \in \mathcal{X}$
$x_{\text{best}} \leftarrow x_{\text{current}}$
$\text{best\_cost} \leftarrow c(x_{\text{best}})$

// Step 2 - Choice and termination
do
Choose a solution $x_{\text{next}} \in \mathcal{N}(x_{\text{current}})$
if the choice criteria cannot be satisfied
by any member of $\mathcal{N}(x_{\text{current}})$,
or the terminating criteria apply
break

// Step 3 - Update
$x_{\text{current}} \leftarrow x_{\text{next}}$
if $c(x_{\text{current}}) < \text{best\_cost}$
$x_{\text{best}} \leftarrow x_{\text{current}}$
$\text{best\_cost} \leftarrow c(x_{\text{best}})$
while (true)

The general structure of a local search algorithm.
Simulated Annealing
Simulated annealing

\[ x_{\text{current}} \in X \quad \text{// an initial solution} \]
\[ t \leftarrow t_0 \quad \text{// an initial temperature } t_0 > 0 \]
\[ \text{// a temperature reduction function } \alpha \]

do
  do
    Select \( x_{\text{next}} \in \mathcal{N}(x_{\text{current}}) \)
    \[ \delta \leftarrow f(x_{\text{next}}) - f(x_{\text{current}}) \]
    if \( \delta < 0 \)
      \( x_{\text{current}} \leftarrow x_{\text{next}} \)
    else
      generate a random number \( p \) in the range \((0,1)\)
      if \( p < e^{-\delta/t} \)
        \( x_{\text{current}} \leftarrow x_{\text{next}} \)
  while \text{iteration\_count} \leq N \]
\[ t \leftarrow \alpha(t) \]
while stopping condition \neq \text{true} \]
return \( x_{\text{current}} \)

The general structure of the simulated annealing algorithm.
Simulated annealing (cont’d)

An example of penalty values for outer loop for graph coloring (451 vertices and 8691 edges).

K. Kuchcinski
Tabu search
Tabu search

- Neighborhood search which uses short-term memory and long-term memory.
- The short-term memory stores tabus to prohibit selection of similar moves unless they fulfill aspiration criteria (prohibit cycling).
- The long-term memory (frequency-based memory) is used to diversify the search.
- When no admissible moves exist a diversification can be performed.
Tabu search algorithm

Step 1 Construct initial configuration $x^{\text{now}} \in \mathcal{X}$

Step 2 for each solution $x_k \in \mathcal{N}(x^{\text{now}})$ do

- Compute change of cost function $\Delta C_k = C(x_k) - C(x^{\text{now}})$

Step 3 for each $\Delta C_k < 0$, in increasing order of $\Delta C_k$ do

- if $\neg \text{tabu}(x_k) \lor \text{tabu\_aspirated}(x_k)$ then
  
  $x^{\text{now}} = x_k$; goto Step 4

- for each solution $x_k \in \mathcal{N}(x^{\text{now}})$ do $\Delta C'_k = \Delta C_k + \text{penalty}(x_k)$

- for each $\Delta C'_k$ in increasing order of $\Delta C'_k$ do

  - if $\neg \text{tabu}(x_k)$ then

    $x^{\text{now}} = x_k$; goto Step 4

Generate $x^{\text{now}}$ by performing the least tabu move

Step 4 if iterations since previous best solution < $Nr\_f\_b$ then goto Step 2

if restarts < $Nr\_r$ then

  Generate new initial configuration $x^{\text{now}}$

  goto Step 2

Step 5 return solution corresponding to the minimum cost function
Tabu search – example
Tabu search – example (cont’d)

![Graph](ts400.dat)
1. The first group of methods is a local search method that uses CP to explore the neighborhood.
   - local search explores the space of solutions,
   - CP is used locally to explore the neighborhood.

2. The second group uses CP as the main search method
   - CP is used to build the search tree,
   - in each node of the search tree LS explores possible solutions.
CP search conclusions

- Search is an important part of constraint programming framework.
- Most systems use some kind of depth-first-search combined with branch-and-bound.
- Local search provides a new paradigm and can be integrated with constraints in different ways.
  - Use local search to select branches in DFS,
  - Use of standard constraint programming search methods for selection of local moves.
  - Use of penalties for constraints which are not satisfied.
  - ...
- Selecting a good search strategy is still an art based on experience.