CSCI5240 - Combinatorial Search and Optimization with Constraints Fall 2015

Teacher: Prof. Jimmy Lee

Assignment 3

Due: 7 December, 2015 (Monday)

1. *The Moving Problem*

   Johnny is the manager of a moving company. As business is improving by the days and Johnny would like to optimize the operations and profits of the company, he decides to automate manpower planning and scheduling. The main factors to consider in such a planning exercise include what major furniture items are to be moved and the manpower (number of movers) and time requirement (minimum moving time) to move each of such items.

   Well, movers are human beings and can get more and more tired during demanding physical activities. In the planning, Johnny must take into account the fact that his crew will move slower and slower after moving more items. A mover can spend k more units of time (called $fatigue\ allowances$) when moving the current item after he has moved k items previously. Now the time used to move an item is increased by the maximum fatigue allowance among all movers moving the item at the moment.

   Johnny wants his crew to finish moving all items as soon as possible, and he comes to you for help. Given a collection of items along with the manpower and time requirement, you are asked to design a schedule of moving these items such that the time used to finish the whole moving job is minimal.

   The description of a job of moving n items with m movers starts with a line that contains the maximum time allowed and the number of movers available. It is then followed by n lines, each of which contains the name, minimum moving time and number of movers needed of the ith item. Your schedule should first output two lines. The first and the second lines contain the start time and the end time of moving each item respectively. They both follow the order of the items used in the input. These two lines are then followed by an $n \times m$ matrix of $\{0, 1\}$ that indicates the assignment of the movers. The jth mover is assigned to move the ith item if the (i, j)-element of the matrix is 1, and not assigned otherwise.

   An example job description is as follows (time is assumed to start from 0):

   ```
   80 4
   piano 30 3
   chair 10 1
   bed 15 3
   table 15 2
   ```

   And the one possible optimal schedule is as follows:

   ```
   0 46 30 46
   30 58 46 62
   0 1 1 1
   0 0 0 1
   1 0 1 1
   1 1 0 0
   ```

   Note that moving the bed takes 1 extra time unit as worker 3 and worker 4 get tired after moving the piano. Similar delays happen in moving the chair and the table.

   Now you are faced with a harder problem:

   ```
   80 5
   cabinet 15 2
   ```

```
piano 30 3
chair 10 1
bed 15 3
table 15 3
```

(a) Give a CSP model of the new problem given above and implement the model using Gecode. Find an optimal solution using the `INT_VAL_MIN` value ordering with the default branching procedure.

(b) Implement the SPLIT branching procedure given on page 63 of the textbook (or to follow the guidelines in tutorial), and use this branching procedure to solve the above problem. Compare the number of fails to find the optimal solution with the default branching procedure.

2. *The Langford's Problem*

A Langford's problem $L(k, n)$ is a permutation problem stated as follows: arrange k sets of numbers 1 to n into a single sequence, such that each appearance of the number m is m numbers on from the last.

The model can be represented by CSP using $n \times k$ variables, $x_1, \ldots, x_{nk}$, each with the domain $[1 \ldots kn]$. Each variables x_{ik+j+1}, where $i \in \{0, \ldots, n-1\}$ and $j \in \{0, \ldots, k-1\}$, denotes the position where the $(j+1)^{th}$ occurrence of the number $i+1$ appears in the sequence.The constraints are:

- `all_different`$(x_1, \ldots, x_{nk})$
- For $i \in \{0, \ldots, n-1\}$, $j \in \{1, \ldots, k-1\}$, $x_{ik+j+1} = x_{ik+j} + i + 2$

Another way to represent this problem as a CSP still uses $n \times k$ variables, $y_1, \ldots, y_{nk}$, each with domain $[0 \ldots kn-1]$. However, each variable y_i denotes the number with its occurrence that appears at the i^{th} position in the sequence. (We denote the number appear at the i^{th} position by $(y_i/k) + 1$ and its occurrence by $(y_i \bmod k) + 1$.) The constraints are:

- `all_different`$(y_1, \ldots, y_{nk})$
- For $i \in \{0, \ldots, n-1\}$, $j \in \{1, \ldots, nk - (k-1)(i+2)\}$, $c \in \{1 \ldots k-1\}$, $y_j = ik$ implies $y_{j+c(i+2)} = ik + c$
- For $i \in \{0, \ldots, n-1\}$, $j \in \{nk - (k-1)(i+2) + 1, \ldots, nk\}$, $y_j \neq ik$

(a) Implement both models using Gecode Solver. Compare the runtime and number of fails to find all possible series for the instances $L(2, 7)$, $L(2, 8)$ and $L(3, 9)$.

(b) *Redundant Modeling* combines multiple models of the same problem using *channeling constraints* to get the advantage of all the models. Each channeling constraint relates the variables between two models. For instance, to combine the two models for the Langford's problem, the following channeling constraints can be used: x_i takes the value of j if and only if y_j takes the value of $i-1$, for $i \in \{1, \ldots, kn\}$ and $j \in \{1, \ldots, kn\}$. Since each model is complete to solve the problem, we have the choice of searching with variables:

- $x_1, \ldots, x_{kn}$ of the first model,
- $y_1, \ldots, y_{kn}$ of the second model, or
- $x_1, \ldots, x_{kn}, y_1, \ldots, y_{kn}$ of both models together.

Write an Gecode program combining the two models using the channeling constraints just described. Implement all three choices of variables for searching. Use `INT_VAR_SIZE_MIN` as variable ordering heuristic for all three choices. Compare the runtime and number of fails to find all possible series for the instances $L(2, 7)$, $L(2, 8)$ and $L(3, 9)$.

(c) Consider the series $(4, 1, 3, 1, 2, 4, 3, 2)$, which is the solution for the instance $L(2, 4)$. We can construct one more symmetric solution $(2, 3, 4, 2, 1, 3, 1, 4)$ easily by reversing the series. We can eliminate this symmetry by adding constraints to reduce the search space. We can simply insist the first occurrence of one is at the first half of the sequence. Add the above symmetry breaking constraint to your combined model and compare the runtime and number of fails to find all possible series for the instances $L(2, 7)$, $L(2, 8)$ and $L(3, 9)$.

3. Recall the GENET local search method for solving binary CSPs. A CSP (Z, D, C) is first translated into a GENET network, in which each variable $x \in Z$ is translated into a cluster of *label nodes* $\langle x, v\rangle$, one for each domain value $v \in D(x)$. Each label node has an *output* (0 or 1) to indicate the node's on-off status. For each binary constraint $c \in C$ involving variables x and y, if $a \in D(x)$ and $b \in D(y)$ violate c, then there is a connection between the corresponding label nodes $\langle x, a\rangle$ and $\langle y, b\rangle$ with a weight initially set to -1. Each label node is associated with an *input*, which is the weighted sum of its connected nodes' output. Initially, one label node per cluster is switched on randomly. Each cluster then updates its on-node by selecting the label node with the maximum input to be on next (or minimizing the "amount" of constraint violation in the network). Clusters can be updated in parallel either synchronously or asynchronously. In *synchronous* update, all clusters calculate their node inputs and perform state update at the *same* time. In *asynchronous* update, each cluster performs input calculation and state update independently.

   Suppose we implement GENET with **synchronous** update. Give a **simple** and **small** CSP and an initial state so that our synchronous GENET would go into state oscillation (or infinite loop), without even settling in a local minimum. Draw the corresponding GENET network of the CSP, and give some execution trace to explain why the execution would end up in an oscillation.

4. Consider the following four types of constraints

   - The **element constraint** mimics the behavior of an array. The constraint
     $$element(I, [V_1, \ldots, V_n], X)$$
     maintains the relationship that if $I = i$, then $X = V_i$, where I, X are variables and V_i's are values.
   - The **global cardinality constraint (gcc)** restricts the appearances of numbers in a sequence of variables. The constraint
     $$gcc([X_1, \ldots, X_n], (V_1, l_1, u_1), \ldots, (V_k, l_k, u_k))$$
     requires that the value V_k appears in $[X_1, \ldots, X_n]$ at least l_k times and at most u_k times. Note that n and k need *not* be equal.
   - The **nvalues constraint** constrains the numbers of distinct values in a sequence of variables. The constraint
     $$nvalues([X_1, \ldots, X_n], k)$$
     imposes that the sequence $[X_1, \ldots, X_n]$ takes exactly k distinct values.
   - The **value precedence constraint** enforces the appearance order of two values. The constraint
     $$precede([X_1, \ldots, X_n], V_1, V_2)$$
     requires if there exists j such that $X_j = V_2$, then there must exist $i < j$ such that $X_i = V_1$.

   Suppose a tuple $\langle h, i, j, k\rangle$ denotes the valuation $\{H \mapsto h, I \mapsto i, J \mapsto j, K \mapsto k\}$, and
   $$\{\langle 3, 4, 2, 4\rangle, \langle 3, 1, 4, 4\rangle, \langle 3, 4, 4, 2\rangle, \langle 3, 4, 1, 4\rangle, \langle 1, 4, 3, 4\rangle, \langle 3, 4, 4, 1\rangle\}$$

is the solution set of a CSP with four variables $\{X_1, X_2, X_3, X_4\}$, where

$$D(X_1) = D(X_2) = D(X_3) = D(X_4) = \{1, 2, 3, 4\}.$$

Suppose the CSP consists of **exactly** `one` constraint of each type listed above. Give the CSP.

5. A linear optimization (minimization) problem is in *basic feasible solved (BFS) form*, if (1) the equations are in solved form, (2) each constant on the right hand side of the equations is non-negative, and (3) only parameters occur in the objective.

   The second stage of the Simplex algorithm assumes that the optimization problem is in BFS form, and consists of repeated execution of the *pivoting step*, which is summarized in the following pseudocode.

```
1.  repeat {
2.      choose a variable $y$ with negative coefficient in the objective function
3.      find the equation $x = b + cy + \cdots$ where $c < 0$ and $-b/c$ is minimal
4.      rewrite this equation with $y$ being the subject $y = -b/c + 1/cx + \cdots$
5.      substitute $-b/c + 1/cx + \cdots$ for $y$ in all other equations and objective function
6.  } until (no such variable $y$ exists) or ($y$ exists but no such equation exists)
7.  if no such $y$ exists then
8.      optimum is found
9.  else
10.     there is no optimal solution
```

   In answering the following questions, use concrete examples to help illustrating your points, and refer to the pseudocode when necessary.

   (a) Explain why the picked equation $x = b + cy + \cdots$ must have $c < 0$ in line 3.

   (b) Explain why the picked equation $x = b + cy + \cdots$ must have a minimal $-b/c$ term among all equations with a negative c in line 3.

   (c) Explain why a variable y of *negative* coefficient should be chosen from the objective in line 2.

   (d) Explain why when there is no longer a variable y with negative coefficient in the objective, the optimal is found.

   (e) Explain why when there is a variable y with negative coefficient in the objective but no equations with a negative c, the optimization problem has no optima in line 10.