UNIVERSITÉ CATHOLIQUE DE LOUVAIN

PROJECT 1 : DISCOVERY OF COMET

LINGI2365 - Constraint Programming

Auteurs :
Vanwelde Romain (3143-10-00)
Crochelet Martin (2236-10-00)

Superviseurs :
Pr. Yves Deville
François Aubry

20 février 2014

Table des matières

1 Nightmare in INGI

1.1 Add the remaining facts to the model

Here are the remaining facts we add to the model :

```
// The killer hates his victim
cp.post(hates[killer , alice] == 1);
// Nobody is taller than himself
forall(p in Person) cp.post(taller[p, p] == 0);
// If A is taller than B, then B is not taller than A
forall(p1 in Person, p2 in Person : p1 != p2) cp.post(taller[p1, p2] == 1
    => taller[p2, p1] == 0);
// The killer is taller than is victim
cp.post(taller[killer ,alice] == 1);
// Bob hates no one that Alice hates
forall(p in Person) cp.post(hates[alice , p] == 1 => hates[bob, p] == 0);
// Alice hates everyone except Chuck -> We verify that p1 is different of
    Alice or Chuck
forall (p1 in Person) cp.post((or (p2 in {alice , chuck}) p1==p2) == 0 =>
    hates[alice ,p1] == 1);
// Chuck hates everyone that Alice hates
forall(p in Person) cp.post(hates[alice ,p] == 1 => hates[chuck ,p] == 1);
//forall(a in Person : hates[alice ,a] == 1) cp.post(hates[chuck ,a] == 0);
    //for question 1.2
// Nobody hates everyone
forall (p1 in Person) cp.post((and(p2 in Person: p2 != p1) hates[p1 ,p2]==1
    ) == 0);
```

facts.co

1.2 Why does the following line result in an execution error ?

Because the problem becomes unsolvable. Indeed, fact (g) allows us to determine that Alice hates at least one person. Then fact (h) determines that this person is also hated by Chuck. But the new line we add for this question determines that the concerned person isn't hated by Chuck. There is thus a contradiction.

1.3 Write the lines you had to add in your report along with who killed Alice

The lines we added are available at Question 1. The Killer computed by the program is Bob.

1.4 Provide the number of solutions returned by your code. Are there different possible killers ?

We use `solveall` to resolve the problem and to determine all the solutions. The counter we placed in the code indicates us that there is only one solution : Bob is the killer.

```
Nb : 1
The killer is: Bob
```

2 Discrete tomography

2.1 Constraints we used

Here are the constraints we used for this problem :

```
forall(i in 1..n) cp.post( sum (j in 1..m) x[i,j] == row[i] );
forall(j in 1..m) cp.post( sum (i in 1..n) x[i,j] == col[j] );
forall(i in 1..(n + m - 1)) cp.post( sum (j in 1..n, k in 1..m : ((j+k) ==
    (i+1)) )  x[j,k] == dia[i] );
```

constraints.co

2.2 Solutions to the 4 instances

We will observe on the bellow solutions that this way of reconstructing the image leads to only one solution for the three first files, but to four different solutions for the last file.

```
Instance #1:
+-------+
|# # ###|
|# #   #|
|# #   #|
|### ###|
|  # #  |
|  # #  |
|  # ###|
+-------+
Number of solutions: 1
```

```
Instance #2:
+--------+
|###     |
|#       |
|#       |
|#    ###|
|###  # #|
|     ###|
|     #  |
|     #  |
+--------+
Number of solutions: 1
```

```
Instance #3:
+---------------------+
|                     |
| # # ### #   #   ### |
| # # #   #   #   # # |
| ### ##  #   #   # # |
| # # #   #   #   # # |
| # # ### ### ### ### |
|                     |
+---------------------+
Number of solutions: 1
```

```
Instance #4:
+--------------------------------+
|                                |
| ###                            |
| #                              |
| #           ###  ###    #  # # |
| #    ###      #  # #   ##  # # |
| ###  # #    ###  # #  # #  ### |
|      ###    #    # #    #    # |
|      #      ###  ###    #    # |
|      #                         |
|                                |
+--------------------------------+
+--------------------------------+
|                                |
| ###                            |
| #                              |
| #      #    # #  ###    #  # # |
| #    #      ###  # #   ##  # # |
| ###  ###     ##  # #  # #  ### |
|      ###    #    # #    #    # |
|      #      ###  ###    #    # |
|      #                         |
|                                |
+--------------------------------+
+--------------------------------+
|                                |
| ###                            |
| #                              |
| #      #    # #  ###    #  # # |
| #    # #    # #  # #   ##  # # |
| ###  # #    ###  # #  # #  ### |
|      ##     ##   # #    #    # |
|      ##      ##  ###    #    # |
|      #                         |
|                                |
+--------------------------------+
+--------------------------------+
|                                |
| ###                            |
| #                              |
| #      #    ##   ###    #  # # |
| #    # #     ##  # #   ##  # # |
| ###  # #    ###  # #  # #  ### |
|      ##     # #  # #    #    # |
|      ##     # #  ###    #    # |
|      #                         |
|                                |
+--------------------------------+
Number of solutions: 4
```

3 NQueens

For each question here, you can find the corresponding code joined in the archive.

3.1 Modify the code of the search procedure such as to assign only values that are still in the domain of the variables.

The step here is to force the domain consistency over the CSP by changing the onBounds parameter to onDomains to force the solver to propagate the consistency into the domains and to verify in the using if the values we try still belong to the domain of the variable. this last part is done with the line : `tryall<m>(val in S : q[queen].memberOf(val))`

3.2 Add a by statement to the forall loop in order to first assign the variables with the smallest number of values in their domains

This heuristic is widely known as the Minimum Remaining Values or MRV and consists of a fail first approach : we constantly choose the variable with the smallest domain in order to try and find rapidly if the current assignment is impossible or not (when the domain size falls to zero).

3.3 Give the search procedure to first assign the queens in the odd columns.

This can be done by replacing the previous ordering function (MRV) by a simpler one : (queen%2) where queen is an unbound variable of the CSP. This forces the search to first bind the queens situated on the odd columns and then the other ones.

3.4 Give the search procedure to first assign the queens in the odd columns and first try to assign values that are less present in the domain of the other variables.

This heuristic is inspired by the MRV one but requires more computation time since we have to compute for each unbound variables the frequency of each remaining possible value of the current queen (queen we want to assign at this iteration).

3.5 Give the search procedure to split (in 2) the domain of an unbound queen.

This heuristic forces to first consider the first values of the domain of the variable over the last ones. This heuristic is here not really indicated since we can't say that the first values are more likely to yield a solution than the other however this heuristic can be really interesting in problems where some values have a more important probability to result in an answer than others.

3.6 Give the search procedure to assign the queens from the sides to the center (so first the queen 1, then queen N, then queen 2,...)

Our choice here is a function that projects the set of queens on to a negative growing list of values that corresponds to the order in which we have to consider them. This function is the following :

$$order = - \mid originalSet - \frac{max(originalSet)}{2} - 1 \mid \tag{1}$$

The minus one is there to make sure that the first queen (column 1) is considered before the last one (column 8) :

$$(- \mid 1 - 4 - 1 \mid = -4) < (-3 = - \mid 8 - 4 - 1 \mid) \tag{2}$$

4 Bin packing

4.1 Explain intuitively what is a domain consistency for a CSP.

The domain consistency for a CSP is a technique that removes for every variables the impossible values for this variable. In opposition to the bound consistency that only removes values that belongs to the boundary of the domain (and that are impossible), the domain consistency will remove **every** value of the domain that violates the constraints in which the variable participates. We end up with an equivalent CSP but much simpler thanks to the smaller domains of its variables. Moreover, if one domain finishes with no possible values then the problem in impossible and we can safely terminate the execution.

4.2 Suppose that we put objects number 2 and 4 into the first bin. What are the domains of the variables after applying domain consistency on this CSP ?

By doing so, we force the weight of the first bin to 9. This implies that no variable of the problem will be able to enter this particular bin since there are no object of weight 1. We have then :

— Item 1 : bins 2 or 3
— Item 2 : bin 1
— Item 3 : bins 2 or 3
— Item 4 : bin 1
— Item 5 : bins 2 or 3
— Item 6 : bins 2 or 3
— Item 7 : bins 2 or 3

4.3 Explain what a redundant constraint is. What are the advantages and disadvantages of such constraints ?

A redundant constraint is a constraint that is a duplicate of another one or a constraint that is a relaxed version of another - meaning a constraint that can be derived from another one. Such a redundant constraint could allow us to approximate the solution using the relaxed constraint and verify it with the "stronger" one. In opposition, a redundant constraint, if treated like any other constraint (without taking advantage of it) generates more non useful computation since we already have the "strong" version of the constraint.

4.4 Add the following constraint to the model : [...]

By adding the new constraint and keeping the problem with the forced assignations of the second point, we make the CSP impossible : the sum of the bins can only be 29 which is different from the sum of the weights which is 30. The programs stops the search of a solution since the maximum weight for each bin is 10 and the weight of the bin 1 is maxed out at 9 due to the assignment.