Cutting-Stock Revisited

H. Simonis
COSYTEC SA
4, rue Jean Rostand
91893 Orsay Cedex
France
Helmut.Simonis@cosytec.com

1 Abstract

In this paper we reconsider a problem solved 10 years ago with CHIP. Using the new search tree tool in CHIP, we analyze the previous results and better explain the behavior of the different programs. The then show how to combine the different finite domain models and how to link in the linear solver in order to improve the constraint reasoning. While this problem is not very hard by current standards, it illustrates the need to define the constraint model properly in order to find solutions quickly. The search tree tool in CHIP is used to understand the search behavior, examples of the search trees generated are shown.

2 Introduction

In [DSV88] [DSV92], we have presented a CLP solution to a two dimensional cutting stock problem originally given in [Cos84]. At this time constraint programming was in an early stage, and we were quite satisfied to come up with an competitive solution to the integer programming model given in [Cos84]. The original paper presented two alternative CHIP models, one using 0/1 finite domain variables, the other using a much smaller set of variables with larger domains together with element constraints. With the CHIP system at that time, the second model proved to be much more efficient. With the CHIP compiler [AB93], the situation was quite different. Using a different implementation of both arithmetic and element constraints, the 0/1 finite domain model suddenly was faster than the element based model, but still used a much larger number of choice points.

In this paper we want to reconsider this problem and its CHIP solution mainly for two reasons:

- The search tree tool for CHIP [SA97], developed in the DISCIPL Esprit project, gives us a much better possibility to understand the search behavior of the programs. We can directly compare different strategies and understand where the backtracking search is spending its time.
- The integration of linear solvers in the CHIP system provides us with an improved constraint reasoning for arithmetic constraints. We want to explain the different ways the solvers can be integrated and used together to find solutions more rapidly.

It should be noted that the problem itself should no longer be considered as particularly difficult. We will show a model with a state-of-the-art integer programming tool, which solves the problem very rapidly. The interest in the problem

lies more in the somewhat changed interpretation of the results of the different models and in the possibility to explain combination techniques which are useful for many, if not most, constraint programs.

3 Problem Description

A detailed description of the problem can be found in [DSV88]. We just recall the main principles. The problem arises in the furniture manufacturing where rectilinear pieces of wood must be cut into smaller pieces. The demand of each type of piece is given, and we try to satisfy the demand with minimal cost. Figure 1 shows the size of the pieces and their demand.

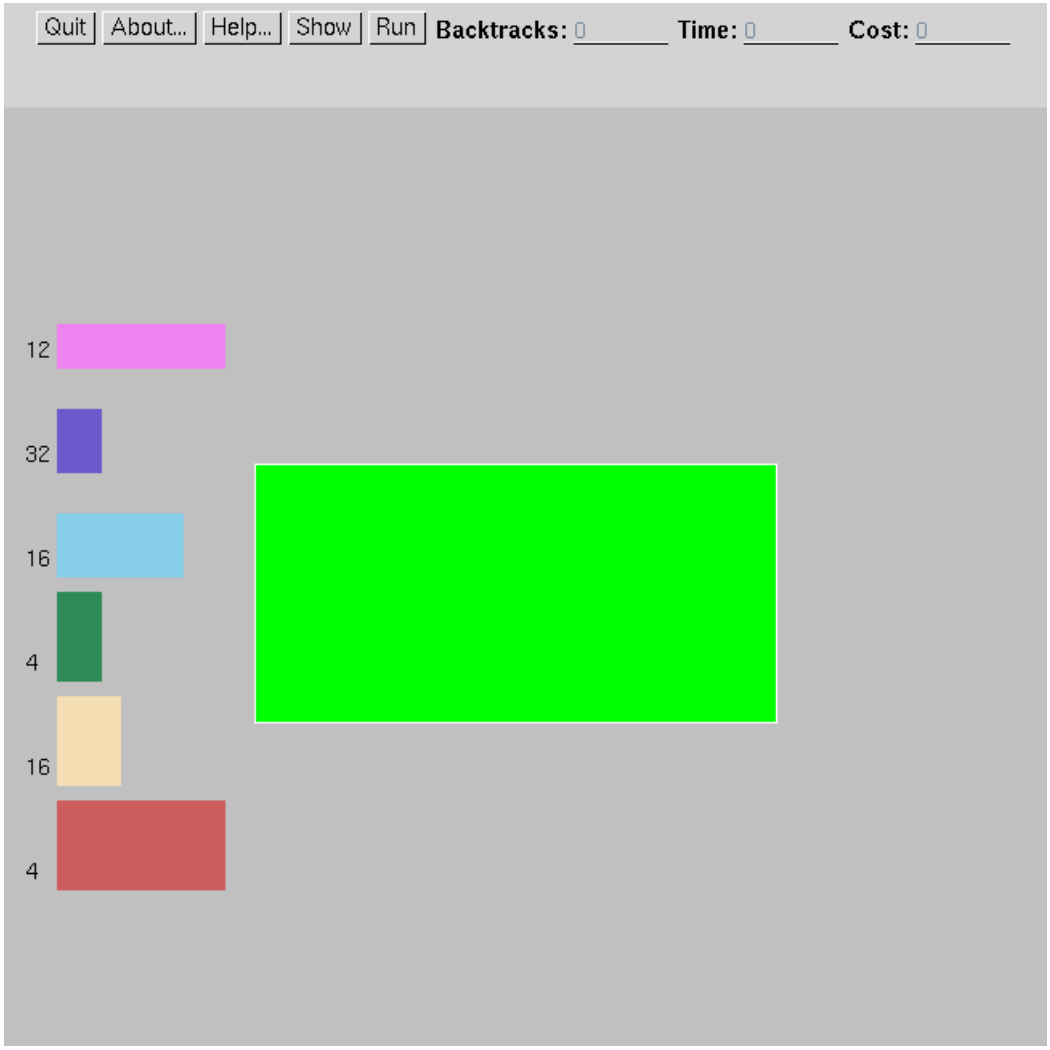


Figure 1: Size and demand of items

Due to the technique of cutting, the number of possible cut combinations is severely limited. The overall approach to solving the problem is to generate all possible cut combinations and then to select a combination of patterns that satisfies the demand. We will here not consider the problem of generating the cut pattern, but figure 2 shows all possible 72 patterns that we have to consider.

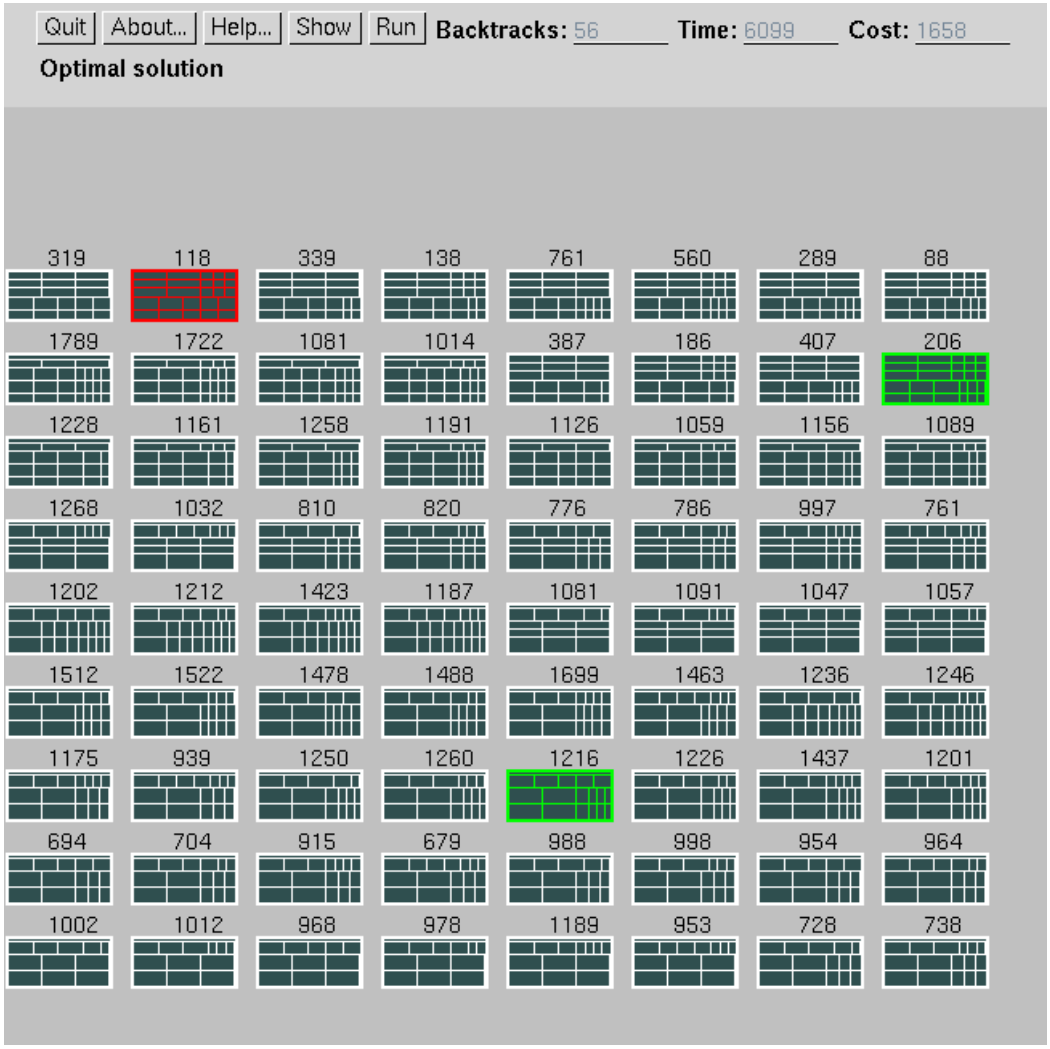


Figure 2: Possible configurations

For each pattern, we now the amount of waste, that is material that is not used for generated pieces. We also know the number of items of different types in each pattern. Given the demand, it is clear that four pattern should be chosen to satisfy the demand. If for some item type, we produce more than the required number, we can use these pieces for stock, and they are not counted as waste. The colored items in figure 2 give an example solution, with the read pattern selected twice and the green pattern selected once each. In order to obtain a simpler constraint model, we may prohibit to use one pattern more than once, but this may change the optimal solution.

4 The two models

In the original paper, we described two models of the problem in CHIP. The first one, similar to the integer programming model used in [Cos84], uses 0/1 finite domain

variables. The second, more in the spirit of finite domain constraint solving, uses a much smaller set of variables which have a larger domain. We now present these two CHIP models.

4.1 0/1 finite domain

This model is quite straightforward and the corresponding CHIP program is given below as program 1. As we have 72 possible pattern, we introduce 72 variables which state if we use the pattern or not. The value 0 denotes that we do not use the pattern, the value 1 states that we use the pattern. As we know that we will use exactly four pattern, we state the sum of the variables is equal to 4. With similar constraints, we state that the demand for each item type must be satisfied. The coefficients of the variables are the number of items produced for each pattern. We can also calculate the total amount of waste by adding up the waste of each pattern that is selected. This means that we can express the complete problem just using linear arithmetic constraints.

```
top(Y,Cost):-
       model_01(Y,Cost),
       min_max(labeling(Y,0,input_order,indomain),Cost).

model_01(Y,Cost):-
       Y = [Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8,Y9,
       Y10,Y11,Y12,Y13,Y14,Y15,Y16,Y17,Y18,Y19,
       Y20,Y21,Y22,Y23,Y24,Y25,Y26,Y27,Y28,Y29,
       Y30,Y31,Y32,Y33,Y34,Y35,Y36,Y37,Y38,Y39,
       Y40,Y41,Y42,Y43,Y44,Y45,Y46,Y47,Y48,Y49,
       Y50,Y51,Y52,Y53,Y54,Y55,Y56,Y57,Y58,Y59,
       Y60,Y61,Y62,Y63,Y64,Y65,Y66,Y67,Y68,Y69,
       Y70,Y71,Y72],

       Y :: 0..1,
       Cost :: 0:100000,

Y1 + Y2 + Y3 + Y4 + Y5 + Y6 + Y7 + Y8 + Y9 + Y10
 + Y11 + Y12 + Y13 + Y14 + Y15 + Y16 + Y17 + Y18 + Y19 + Y20
 + Y21 + Y22 + Y23 + Y24 + Y25 + Y26 + Y27 + Y28 + Y29 + Y30
 + Y31 + Y32 + Y33 + Y34 + Y35 + Y36 + Y37 + Y38 + Y39 + Y40
 + Y41 + Y42 + Y43 + Y44 + Y45 + Y46 + Y47 + Y48 + Y49 + Y50
 + Y51 + Y52 + Y53 + Y54 + Y55 + Y56 + Y57 + Y58 + Y59 + Y60
 + Y61 + Y62 + Y63 + Y64 + Y65 + Y66 + Y67 + Y68 + Y69 + Y70
 + Y71 + Y72 #= 4,
Cost #= 1002 * Y1 + 1012 * Y2 + 968 * Y3 + 978 * Y4 + 1189 * Y5
 + 953 * Y6 + 728 * Y7 + 738 * Y8 + 694 * Y9 + 704 * Y10 + 915 * Y11
 + 679 * Y12 + 988 * Y13 + 998 * Y14 + 954 * Y15 + 964 * Y16 + 1175 *
Y17 + 939 * Y18 + 1250 * Y19 + 1260 * Y20 + 1216 * Y21 + 1226 * Y22
+ 1437 * Y23 + 1201 * Y24 + 1512 * Y25 + 1522 * Y26 + 1478 * Y27 + 1488 *
Y28 + 1699 * Y29 + 1463 * Y30 + 1236 * Y31 + 1246 * Y32 + 1202 *
Y33 + 1212 * Y34 + 1423 * Y35 + 1187 * Y36 + 1081 * Y37 + 1091 * Y38
+ 1047 * Y39 + 1057 * Y40 + 1268 * Y41 + 1032 * Y42 + 810 * Y43 + 820
* Y44 + 776 * Y45 + 786 * Y46 + 997 * Y47 + 761 * Y48 + 1228 * Y49 +
 1161 * Y50 + 1258 * Y51 + 1191 * Y52 + 1126 * Y53 + 1059 * Y54 + 1156
 * Y55 + 1089 * Y56 + 1789 * Y57 + 1722 * Y58 + 1081 * Y59 + 1014 * Y60 +
387 * Y61 + 186 * Y62 + 407 * Y63 + 206 * Y64 + 319 * Y65 + 118
* Y66 + 339 * Y67 + 138 * Y68 + 761 * Y69 + 560 * Y70 + 289 * Y71 +
88 * Y72,
6 * Y1 + 6 * Y2 + 6 * Y3 + 6 * Y4 + 6 * Y5 + 6
 * Y6 + 4 * Y7 + 4 * Y8 + 4 * Y9 + 4 * Y10 + 4 * Y11 + 4 * Y12 + 4
 * Y13 + 4 * Y14 + 4 * Y15 + 4 * Y16 + 4 * Y17 + 4 * Y18 + 4 * Y19
 + 4 * Y20 + 4 * Y21 + 4 * Y22 + 4 * Y23 + 4 * Y24 + 4 * Y25 + 4 *
Y26 + 4 * Y27 + 4 * Y28 + 4 * Y29 + 4 * Y30 + 2 * Y31 + 2 * Y32 + 2
 * Y33 + 2 * Y34 + 2 * Y35 + 2 * Y36 + 3 * Y37 + 3 * Y38 + 3 * Y39
 + 3 * Y40 + 3 * Y41 + 3 * Y42 + 2 * Y43 + 2 * Y44 + 2 * Y45 + 2 *
```

```
Y46 + 2 * Y47 + 2 * Y48 #>= 4,
1 * Y1 + 3 * Y3 + 2 * Y4 + 1 * Y5 +
3 * Y6 + 7 * Y7 + 6 * Y8 + 9 * Y9 + 8 * Y10 + 7 * Y11 + 9 * Y12 +
5 * Y13 + 4 * Y14 + 7 * Y15 + 6 * Y16 + 5 * Y17 + 7 * Y18 + 3 * Y19
 + 2 * Y20 + 5 * Y21 + 4 * Y22 + 3 * Y23 + 5 * Y24 + 1 * Y25 + 3 *
Y27 + 2 * Y28 + 1 * Y29 + 3 * Y30 + 7 * Y31 + 6 * Y32 + 9 * Y33 +
8 * Y34 + 7 * Y35 + 9 * Y36 + 1 * Y37 + 3 * Y39 + 2 * Y40 + 1 * Y41
 + 3 * Y42 + 4 * Y43 + 3 * Y44 + 6 * Y45 + 5 * Y46 + 4 * Y47 + 6 *
Y48 + 3 * Y49 + 3 * Y50 + 9 * Y53 + 9 * Y54 + 6 * Y55 + 6 * Y56 +
3 * Y57 + 3 * Y58 + 9 * Y59 + 9 * Y60 + 2 * Y61 + 2 * Y62 + 6 * Y65
 + 6 * Y66 + 4 * Y67 + 4 * Y68 + 2 * Y69 + 2 * Y70 + 6 * Y71 + 6 *
Y72 #>= 16,
2 * Y13 + 2 * Y14 + 2 * Y15 + 2 * Y16 + 2 * Y17 + 2 * Y18 + 4 * Y19 + 4 *
Y20 + 4 * Y21 + 4 * Y22 + 4 * Y23 + 4 * Y24 + 6
* Y25 + 6 * Y26 + 6 * Y27 + 6 * Y28 + 6 * Y29 + 6 * Y30 + 6 * Y31
+ 6 * Y32 + 6 * Y33 + 6 * Y34 + 6 * Y35 + 6 * Y36 #>= 4,
3 * Y1 + 3
* Y2 + 2 * Y3 + 2 * Y4 + 2 * Y5 + 1 * Y6 + 3 * Y7 + 3 * Y8 + 2 *
Y9 + 2 * Y10 + 2 * Y11 + 1 * Y12 + 3 * Y13 + 3 * Y14 + 2 * Y15 + 2
* Y16 + 2 * Y17 + 1 * Y18 + 3 * Y19 + 3 * Y20 + 2 * Y21 + 2 * Y22
+ 2 * Y23 + 1 * Y24 + 3 * Y25 + 3 * Y26 + 2 * Y27 + 2 * Y28 + 2 * Y29 + 1 *
Y30 + 3 * Y31 + 3 * Y32 + 2 * Y33 + 2 * Y34 + 2 * Y35 + 1
* Y36 + 3 * Y37 + 3 * Y38 + 2 * Y39 + 2 * Y40 + 2 * Y41 + 1 * Y42
+ 3 * Y43 + 3 * Y44 + 2 * Y45 + 2 * Y46 + 2 * Y47 + 1 * Y48 + 9 * Y49 + 9 *
Y50 + 9 * Y51 + 9 * Y52 + 6 * Y53 + 6 * Y54 + 6 * Y55 + 6
* Y56 + 6 * Y57 + 6 * Y58 + 3 * Y59 + 3 * Y60 + 6 * Y61 + 6 * Y62
+ 6 * Y63 + 6 * Y64 + 4 * Y65 + 4 * Y66 + 4 * Y67 + 4 * Y68 + 4 * Y69 + 4 *
Y70 + 2 * Y71 + 2 * Y72 #>= 16,
1 * Y1 + 3 * Y2 + 2 * Y4 +
 3 * Y5 + 3 * Y6 + 1 * Y7 + 3 * Y8 + 2 * Y10 + 3 * Y11 + 3 * Y12 +
 1 * Y13 + 3 * Y14 + 2 * Y16 + 3 * Y17 + 3 * Y18 + 1 * Y19 + 3 * Y20 + 2 *
Y22 + 3 * Y23 + 3 * Y24 + 1 * Y25 + 3 * Y26 + 2 * Y28 + 3 *
 Y29 + 3 * Y30 + 1 * Y31 + 3 * Y32 + 2 * Y34 + 3 * Y35 + 3 * Y36 +
 1 * Y37 + 3 * Y38 + 2 * Y40 + 3 * Y41 + 3 * Y42 + 7 * Y43 + 9 * Y44 + 6 *
Y45 + 8 * Y46 + 9 * Y47 + 9 * Y48 + 3 * Y49 + 6 * Y50 + 9 *
 Y51 + 12 * Y52 + 3 * Y54 + 6 * Y55 + 9 * Y56 + 9 * Y57 + 12 * Y58
 + 9 * Y59 + 12 * Y60 + 2 * Y61 + 11 * Y62 + 6 * Y63 + 15 * Y64 + 9
* Y66 + 4 * Y67 + 13 * Y68 + 6 * Y69 + 15 * Y70 + 6 * Y71 + 15 * Y72 #>= 32,
6 * Y37 + 6 * Y38 + 6 * Y39 + 6 * Y40 + 6 * Y41 + 6 * Y42
 + 4 * Y43 + 4 * Y44 + 4 * Y45 + 4 * Y46 + 4 * Y47 + 4 * Y48 + 3 *
Y49 + 2 * Y50 + 3 * Y51 + 2 * Y52 + 3 * Y53 + 2 * Y54 + 3 * Y55 + 2
 * Y56 + 3 * Y57 + 2 * Y58 + 3 * Y59 + 2 * Y60 + 9 * Y61 + 6 * Y62
 + 9 * Y63 + 6 * Y64 + 9 * Y65 + 6 * Y66 + 9 * Y67 + 6 * Y68 + 9 *
Y69 + 6 * Y70 + 9 * Y71 + 6 * Y72 #>= 12.
```

Program 1: 0/1 finite domain model

4.1.1 Extension to handle doublets

The model above uses 0/1 variables, we can not express the fact that one pattern should be used more than once. To overcome this problem, we can just extend the domains to range from 0 to 4. This added flexibility naturally comes at the cost of a much larger search space, that we will have to explore.

4.2 Finite domain

The finite domain model looks at the problem in the inverse way. Instead of deciding whether we use a pattern or not, we decide which pattern to use. This means that we have four variables which range of a domain of 1 to 72. For each variable an element constraint expresses the waste depending on the selected pattern. The total waste is equal to the sum of these cost variables. In a similar way, the demand constraints are expressed with element constraints and inequality constraints.

The program 2 below shows the actual CHIP program. For the moment, we ignore the third argument of the predicate model, it will be used later on.

```
top(X,Cost):-
       model(X,Cost,Terms),
       min_max(labeling(X,0,input_order,indomain),Cost).

model([X1,X2,X3,X4],Cost,[t(1,X1,C1),t(2,X2,C2),t(3,X3,C3),t(4,X4,C4)]) :-

       [X1,X2,X3,X4] :: 1..72,
       Cost :: 0:100000,
L=
[1002,1012,968,978,1189,953,728,738,694,704,915,679,988,998,954,
964,1175,939,1250,1260,1216,1226,1437,1201,1512,1522,1478,1488,
1699,1463,1236,1246,1202,1212,1423,1187,1081,1091,1047,1057,1268,
1032,810,820,776,786,997,761,1228,1161,1258,1191,1126,1059,1156,
1089,1789,1722,1081,1014,387,186,407,206,319,118,339,138,761,560,
289,88],

QL1 =
[6,6,6,6,6,6,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,2,2,2,2,2,2,3,
3,3,3,3,3,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

QL2 =
[1,0,3,2,1,3,7,6,9,8,7,9,5,4,7,6,5,7,3,2,5,4,3,5,1,0,3,2,1,3,7,6,9,8,7,9,1,
0,3,2,1,3,4,3,6,5,4,6,3,3,0,0,9,9,6,6,3,3,9,9,2,2,0,0,6,6,4,4,2,2,6,6],

QL3 =
[0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,4,4,4,4,4,4,6,6,6,6,6,6,6,6,6,6,6,6,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

QL4 =
[3,3,2,2,2,1,3,3,2,2,2,1,3,3,2,2,2,1,3,3,2,2,2,1,3,3,2,2,2,1,3,
3,2,2,2,1,3,3,2,2,2,1,9,9,9,9,6,6,6,6,6,6,3,3,6,6,6,6,4,4,4,4,4,4,2,2],

QL5 =
[1,3,0,2,3,3,1,3,0,2,3,3,1,3,0,2,3,3,1,3,0,2,3,3,1,3,0,2,3,3,1,3,0,2,3,3,1,
3,0,2,3,3,7,9,6,8,9,9,3,6,9,12,0,3,6,9,9,12,9,12,2,11,6,15,0,9,4,13,6,15,6,1
5],

QL6 =
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6,
6,6,6,6,6,4,4,4,4,4,4,3,2,3,2,3,2,3,2,3,2,3,2,9,6,9,6,9,6,9,6,9,6,9,6],

element(X1,L,C1),
element(X2,L,C2),
element(X3,L,C3),
element(X4,L,C4),

Cost #= C1 + C2 + C3 + C4,

element(X1,QL1,Q11),
element(X2,QL1,Q12),
element(X3,QL1,Q13),
element(X4,QL1,Q14),
element(X1,QL2,Q21),
element(X2,QL2,Q22),
element(X3,QL2,Q23),
element(X4,QL2,Q24),
element(X1,QL3,Q31),
element(X2,QL3,Q32),
element(X3,QL3,Q33),
element(X4,QL3,Q34),
element(X1,QL4,Q41),
element(X2,QL4,Q42),
element(X3,QL4,Q43),
element(X4,QL4,Q44),
```

```
element(X1,QL5,Q51),
element(X2,QL5,Q52),
element(X3,QL5,Q53),
element(X4,QL5,Q54),
element(X1,QL6,Q61),
element(X2,QL6,Q62),
element(X3,QL6,Q63),
element(X4,QL6,Q64),

Q11 + Q12 + Q13 + Q14 #>= 4,
Q21 + Q22 + Q23 + Q24 #>= 16,
Q31 + Q32 + Q33 + Q34 #>= 4,
Q41 + Q42 + Q43 + Q44 #>= 16,
Q51 + Q52 + Q53 + Q54 #>= 32,
Q61 + Q62 + Q63 + Q64 #>= 12.
```

Program 2: finite domain model

4.2.1 Excluding doublets

In order to compare the two models, we should add a constraint which forbids that a pattern is used more than once. In the model above, variables X1 and X2 for example can take the same value. To exclude this, we just add an alldifferent constraint between the decision variables [X1,X2,X3,X4].

```
alldifferent([X1, X2, X3, X4]),
```

Program 3: removing doublets

4.2.2 Removing symmetry

The finite domain model also contains symmetrical solutions, as any permutation of a solution is again a solution with the same cost. If we don't remove this symmetry, we will find the same solution several times, basically wasting computational effort.
To remove the symmetry, we can enforce an order of the decision variables by adding inequality constraints.

```
X1 #< X2,
X2 #< X3,
X3 #< X4,
```

Program 4: removing symmetry

Instead of removing the symmetry on the decision variables, we may want to impose constraints on the cost variables. This will help to impose a lower bound on the total cost by stating the inequalities

```
C1 #<= C2,
C2 #<= C3,
C3 #<= C4,
```

Program 5: symmetry removal on the cost variable

Note that we have to use smaller or equal constraints in this case, as the costs of two different pattern may be the same. Since we do not use the strict inequality, we may still consider some symmetrical solutions. This overhead will be balanced against the good lower bound cost propagation that this form of symmetry removal offers.

4.2.3 Dynamic variable selection

For many CHIP programs, we can improve the search behavior by dynamically selecting variables to assign according to some selection criteria, like the first-fail principle. In this case, as we only have 4 decision variables and we already introduced additional constraints to break the symmetry between them, a static ordering will be sufficient.

4.2.4 Labeling with cost

Another standard method in the search strategy consists in not labeling the decision variables themselves, but first to assign the cost variables and them the decision variables. This way, the decision variables will be selected in order of increasing cost, which is a good strategy to find solutions with small overall cost. Program below shows the CHIP program.

```
top(X,Cost):-
       model(X,Cost,Terms),
       min_max(labeling(Terms,2,input_order,assign),Cost).

assign(t(N,X,C)):-
       indomain(C ),
       indomain(X).
```

Program 6: labeling on cost variables

4.2.5 Indomain1

The built-in indomain predicate tests each value independently. If the instantiation of a value fails, the system backtracks and tries the next alternative. It does not remove the failed value from the domain before continuing. Usually, this is a good strategy, but sometimes, in particular if cost variables are involved, the previous value should be removed from the domain. This is done in the indomain1 predicate, which adds a disequality constraint after each failure as shown in figure 3.

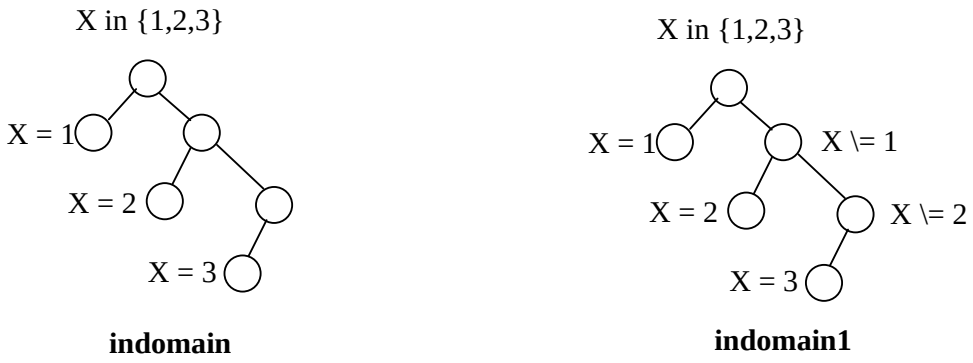


Figure 3: Difference between indomain and indomain1

4.2.6 Domain splitting

To reduce the number of choices even further, we can apply a domain splitting strategy instead of an indomain assignment. At each step of the search we assign one variable. But instead of testing each value independently, we can split the domain into two equal parts by introducing an inequality constraint. After this branching, we

further reduce the domain of that variable until it is instantiated. By splitting the domain, we hope to avoid enumeration of values which are not consistent with the constraint store. The program for the domain splitting is given below.

```
top(X,Cost):-
       model(X,Cost,Cost_variables),
       min_max(labeling(Cost_variables,0,input_order,split),Cost).

split(X):-
       integer(X),
       !.
split(X):-
       domain_info(X,Min,Max,_,_,_),
       Mid is (Min+Max)/2,
       split(X,Mid).

split(X,Mid):-
       X #<= Mid,
       split(X).
split(X,Mid):-
       X #> Mid,
       split(X).
```

Program 7: Domain splitting strategy

Note that we use the splitting operation on the cost variables, not on the decision variables. In this way the added inequality constraint will have an immediate effect on the total cost.

5 A clear winner?

We will now look at the results of the different programs and their variants. The exeuction times are for CHIP V5.2 on a Pentium MMX 266MHz, 64 Mb memory under WindowsNT 4.0.

Program	solutions found	nodes in tree	time (sec)
0/1 model	8	6867	2.84
0/4 model	9	4978	2.56
1..72 model	5	166	3.18
without doublets	4	152	2.95
symmetry on X	4	51	1.20
symmetry on C	7	112	1.98
labeling on cost, symmetry on X	2	27	0.55
labeling on cost, symmetry on C	2	26	0.46
labeling on cost, using indomain1	2	26	0.24
labeling on cost, domain splitting	2	26	0.05

The figure 4 shows the search tree generated by the 0/1 model, figure 5 shows the basic 1..72 domain model, and figure 6 shows the solution obtained with a labeling on the cost variables and symmetry removal on C.

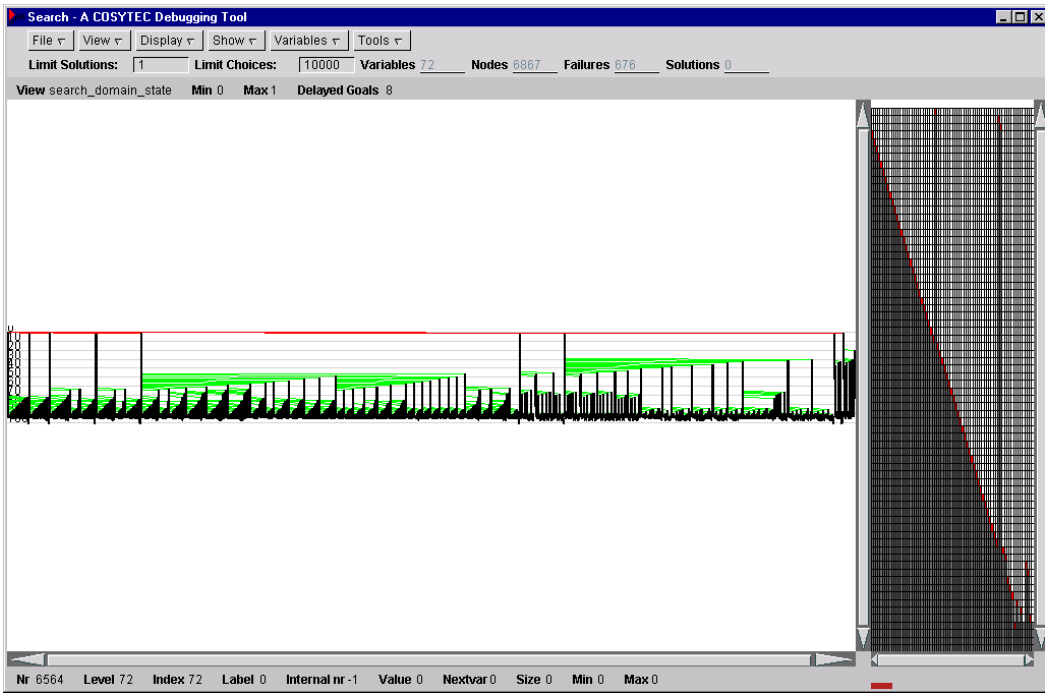


Figure 4: Search space of 0/1 model

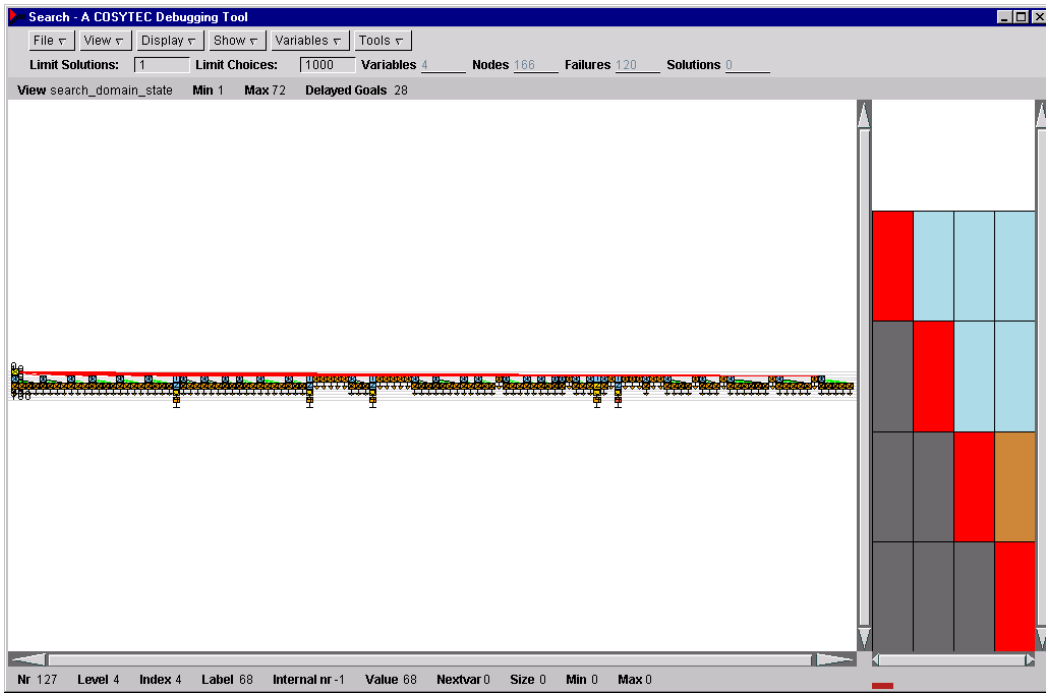


Figure 5: Search space of 1..72 model

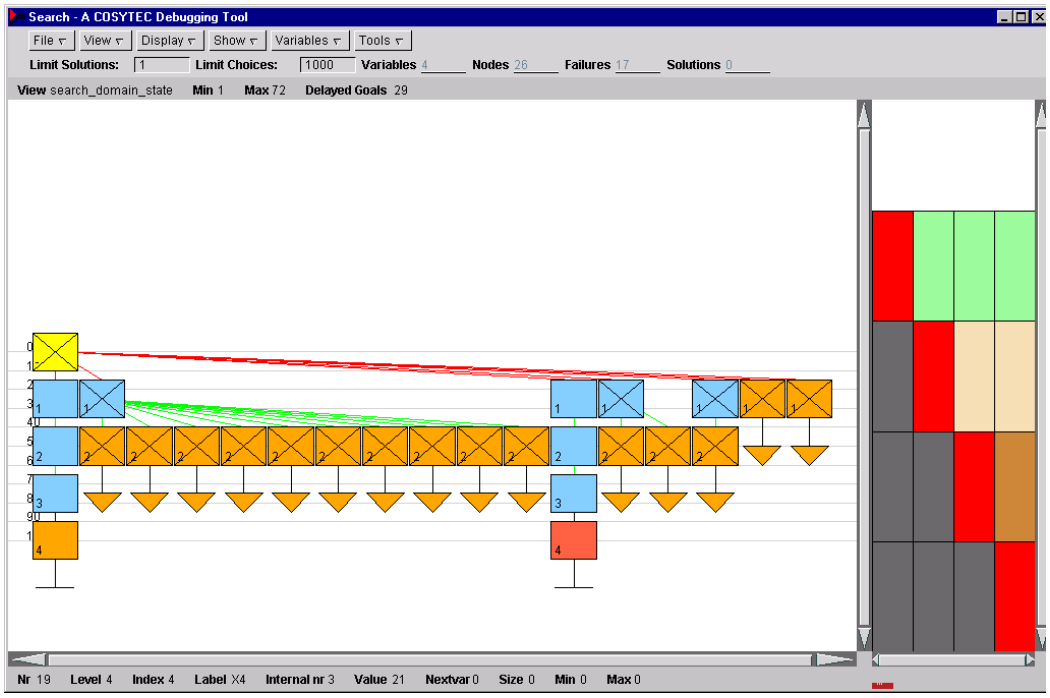


Figure 6: Labeling on Cost

Based on these results, a comparison of the two models is simple. The 0/1 model with finite domains generate 300 times more nodes than the best 1..72 model. Extending the 0/1 model to domains 0..4 leads to a slightly better result, as solutions are found in a different sequence. Clearly the increase of the search space does not lead to a corresponding increase of the search time.

The difference between the worst and the best 1..72 model is a factor of 6. It is clearly worthwhile to remove symmetries and to exclude possible solutions. In this example, it is not clear that removing symmetries on the cost is superior to removal on the decision variables.

The use of indomain1 reduces the number of values that are tested and which fail immediately. The number of nodes in the search tree does not change, but execution time is split by half.

The domain splitting variant again dramatically reduces the execution time. The reason for this is that the system does not have to test individual values separately, but can remove a large number of alternatives in one test. This improvement is not visible in the search tree tool, as immediate failures are not shown.

The difference between the best and worst execution times for the 1..72 model is a factor of 60. This clearly shows the need for defining the constraints and in particular the labeling strategy carefully.

6 Improving the 0/1 model

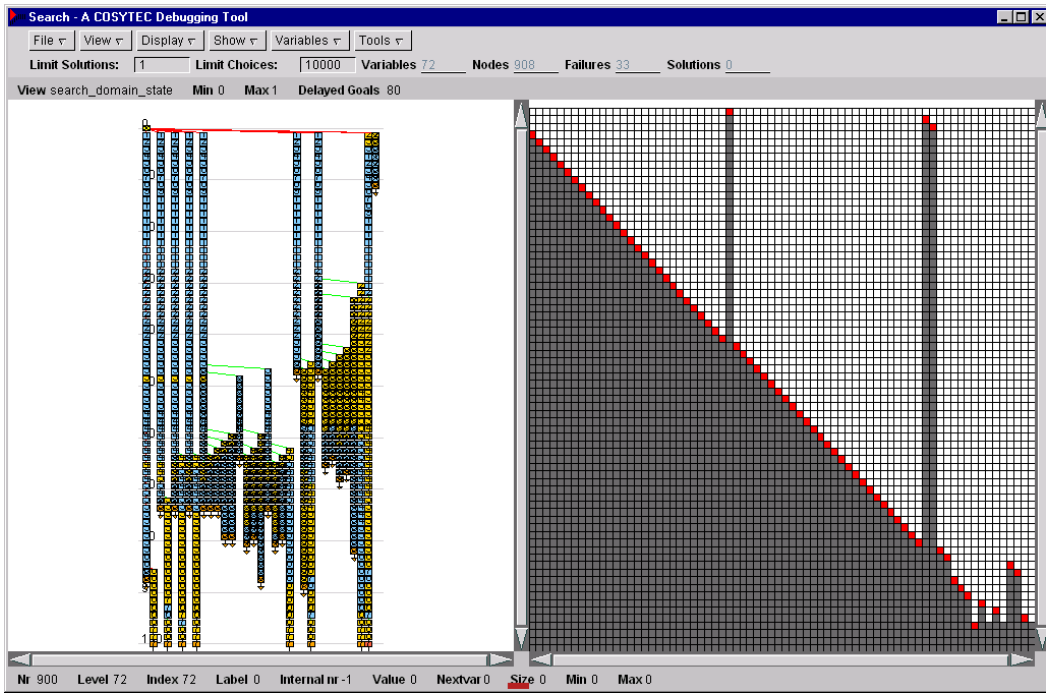
In which way can we improve the 0/1 model? As the domains consist of only two values, we can not easily devise dynamic variable selection or assignment strategies. On the constraint side, we can combine all arithmetic constraint in a linear model. The Simplex algorithm (respectively the Gaussian elimination) keeps ensures consistency of the constraints with the partial assignment. Expressing the linear model in CHIP is quite simple. Instead of defining the variables as domain variables, we create them as

linear variables within the range 0 to 1. We use the linear constraint equality symbols (^>=, ^=) to express the constraints. But how do we link the finite domain variables and the variables of the linear relaxation?
The easiest way is a co-routine which links the two sets of decision variables. As soon as one of them is instantiated, it is bound to the other variable. As the search is performed in the finite domain model, more information is transferred from the finite domain model to the linear model, but the inverse information is important to reduce the search space. The program below shows the interaction.

```
top(X,Cost):-
       model_01(X,Cost,Terms),
       linear_model(Y,Lcost),
       mix(X,Y),
       connect(Cost,Lcost),
       min_max(labeling(X,0,input_order,indomain),Cost).

mix([],[]).
mix([X|X1],[Y|Y1]):-
       mix1(X,Y),
       mix1(Y,X),
       mix(X1,Y1).

?-delay mix1(ground,any).
mix1(X,X).
```



Program 8: Improving the 0/1 model

Figure 7: Improved 0/1 model with linear solver

The link between the two cost variables is only performed in one direction. If a new bound on the cost variable of the finite domain model is stated, this bound is transferred to the linear model. In CHIP, this connection is done with a touched demon, shown in the program below.

```
connect(C,Cr):-
       touched(update,C,Cr,max).

update(C,Cr):-
       domain_info(C,Min,Max,_,_,_),
       Cr ^<= Max.
```

Program 9: Connecting the cost variables

Program	solutions found	nodes in tree	time
0/1 model with linear model	8	908	-[1]

The number of nodes in the tree is reduced by a factor of 7, while the number of solution stays the same. This clearly shows the improved constraint propagation in this model, although it is still performing much more search than the 1..72 model.

7 Combining the models

The next step in improving the program would consist in combining the two finite domain models. Perhaps there is some propagation in one that is not performed in the other? We will see that this is not the case (in fact, we can show that this should not happen given the propagation methods used), but it is a first step to including the linear model in the 1..72 finite domain model.

7.1 How to express the link

We now have the problem how to link 0/1 domain variables with the four decision variables of the other model. A delay coroutine will only perform the connection when the variable is bound. This is not strong enough for our purpose. While we could write a special routine, for example as a touched demon, to perform this task, it is easier to use some of the existing CHIP constraint for the purpose. There are two possibilities, one using the element constraint, the other using the among constraint.

7.1.1 Element constraints

The element constraint in CHIP V5.2 allows a list of domain variables in the place of the cost table. We can link one 1..72 decision variable with 72 0/1 variables in one element constraint, and need four constraints for all decision variables.

```
top([X1,X2,X3,X4],Cost):-
       model([X1,X2,X3,X4],Cost,Terms),
       model_01(Y,Lcost),
       length(Zero,72),
       Zero :: 0..0,
       element(X1,Y,Zero,1,[all,all,all,all]),
       element(X2,Y,Zero,1,[all,all,all,all]),
       element(X3,Y,Zero,1,[all,all,all,all]),
       element(X4,Y,Zero,1,[all,all,all,all]),
       min_max(labeling(Terms,2,input_order,assign),[Cost,Lcost]).
```

Program 10: Combination with element constraints

Unfortunately, this combination is also quite weak. If a value is removed from the domain of one of the Xi, the corresponding Yi variable is not set to zero. This is

[1] Due to on-going work of the integration, no execution times are given for the linear model

actually intended in this case, since the other Xk variable could take that value, but the element constraint is not able to propagate this even in the case that the sum of all Ym is equal to one.

7.1.2 Among constraint

But there is another way to express the link. The among constraint can be used to count which values are used in the four decision variables. The counting variables are our 0/1 variables, and the counted values come from the 1..72 domain variables. As a further simplification, we can use the multiple-among version to express the link for all values in one constraint. This leads to the following program:

```
top(X,Cost):-
       model(X,Cost,Terms),
       model_01(Y,Lcost),
       prepare_values(1,72,Values),
       among(Y,X,[0,0,0,0],Values,all),
       min_max(labeling(Terms,2,input_order,assign),[Cost,Lcost]).

prepare_values(N,M,[]):-
       N > M.
prepare_values(N,M,[[N]|R]):-
       N =< M,
       N1 is N+1,
       prepare_values(N1,M,R).
```

Program 11: Combination with among constraints

This link is much stronger than the connection with the element constraint. As soon as some value is removed from the domain all Xi variables, the corresponding Yi variable will be set to 0.
But when testing this interface, we note that there is no improvement in the constraint propagation over the best 1..72 model. On the other hand, the execution increase significantly due to the solving of both models and the connection constraints. The among constraint, doing more propagation, incurs a higher overhead than the element constraint.

Program	solutions found	nodes in tree	time
combined model (element)	2	26	0.84
combined model (among)	2	26	1.93

7.2 Adding the linear solver

As we now have a connection between the 1..72 model and the 0/1 finite domain model, as well as a connection between the 0/1 finite domain model and its linear relaxation, it is easy connect all three. This connection decreases the number of nodes in the search space, but is significantly more expensive in terms of execution time.

Program	solutions found	nodes in tree	time
combination with linear solver	2	13	-

The search tree of this combined model again is significantly smaller as shown in figure 8:

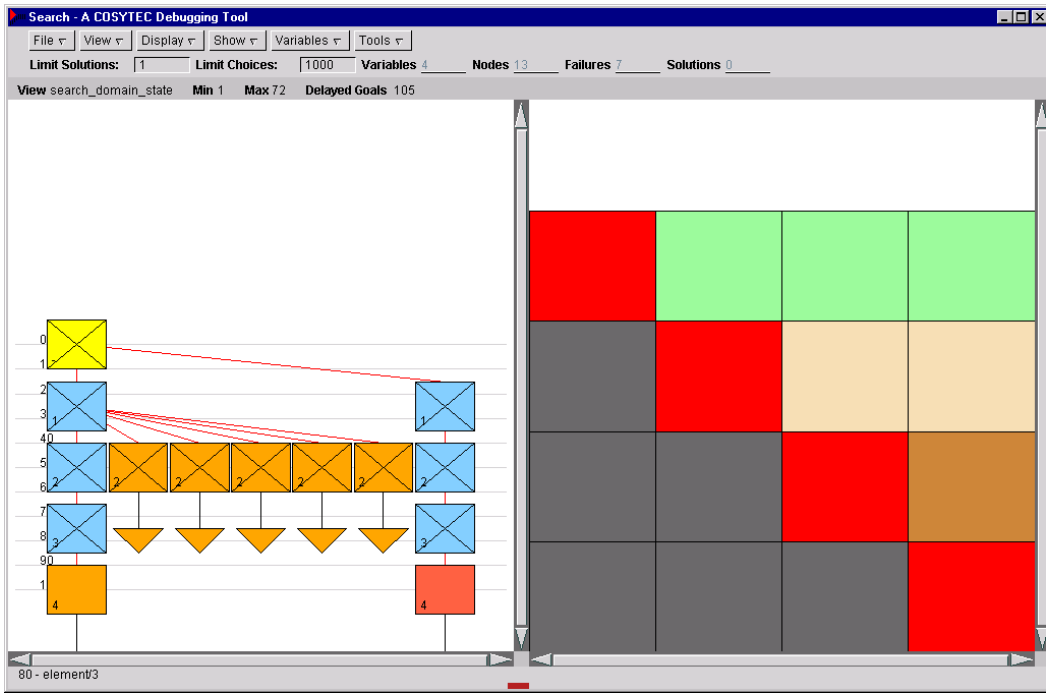


Figure 8: Combined model

We can see from this result that the constraint propagation is actually much better in the linear model. The advantage of the 1..72 model lies in the possibility to

- make the right choices at the right time
- make choices of larger granularity
- create a narrow search tree

8 Going back to the strategy

But can we not simulate the search strategy of the 1..72 model in the 0/1 model as well. We can for example sort the list of 0/1 domain variables according to their cost values and then label the decision variables in that order. We have tested two variants, one sorting the variables in increasing cost order and starting the enumeration with the value 1, or to sort the items in decreasing cost order and starting the enumeration with 0.
Both variants are disappointing, even if we use the linear model in addition:

Program	solutions found	nodes in tree	time
increasing cost	2	22438	12.37
decreasing cost	2	19820	11.57
increasing cost linear model	2	2890	-
decreasing cost linear model	2	590	-

This shows that the advantage of the 1..72 model not only lies in the ability to use the costs as guidance for the search, but also depends on the granularity of the decisions.

9 Conclusions

Let us summarize the results of the different steps in two tables, one for the 0/1 models, and one for the 1..72 models:

0/1 model	8	6867	2.84
0/4 model	9	4978	2.56
0/1 model with linear model	8	908	-
increasing cost	2	22438	12.37
decreasing cost	2	19820	11.57
increasing cost linear model	2	2890	-
decreasing cost linear model	2	590	-

The range of results is quite surprising and shows the need for experimentation in developing CLP programs. The best solution still uses a large number of nodes, compared to the 1..72 models.

Program	solutions found	nodes in tree	time
1..72 model	5	166	3.18.
without doublets	4	152	2.95
symmetry on X	4	51	1.20
symmetry on C	7	112	1.98
labeling on cost, symmetry on X	2	27	0.55
labeling on cost, symmetry on C	2	26	0.46
labeling on cost, indomain1	2	26	0.24
labeling on cost, domain splitting	2	26	0.05
combined model (element)	2	26	0.84
combined model (among)	2	26	1.93
combination with linear solver	2	13	-

Even the worst of the solutions explores less nodes than the best of the 0..1 models. The best solution explores only 13 nodes, using the linear model to remove inconsistent assignments.

10 Summary

In this paper we have discussed different way to model a cutting stock problem in CHIP. Based on a previous paper, we have described 0/1 variable models and a model

with larger domains. We have also discussed the inclusion and integration of a linear model in the finite domain program and the resulting changes in the search space. We obtained the following lessons:

- Adding linear constraints on 0/1 variables linked to a finite domain model may significantly reduce the number of nodes in the search tree. The better search behavior of the finite domain model is not due to better constraint propagation, but to better control over the search routines.
- Using the same strategy, but with variants of an assignment methods, we obtain large reductions in the execution time (factor 10), even though the search tree does not show any difference. This improvement is entirely due to the reduction of value tests which immediately fail in the propagation. Conventional instrumentation does not show this overhead.
- The size of the search tree is only an approximation of the effort it takes to find a solution. The number of nodes is not directly connected to the search time, but also depends on the number of variables, the number of constraints and the propagation methods used.
- The among constraint provides an elegant method to link finite domain decision variables of one model with 0/1 finite domain variables of another model of the same problem.
- Even for a relatively simple problem, the success of a finite domain constraint model depends on a number of choices about models and search strategies.

11 Bibliography

[AB93] A. Aggoun, N. Beldiceanu
Overview of the CHIP Compiler System. in F. Benhamou, A. Colmerauer (Eds), Constraint Logic Programming – Selected Research, MIT Press, 1993

[Cos84] M. C. Costa Une Etude Practique de Decoupes de Panneaux de Bois.
RAIRO, Recherche Operationelle 18 (3), 211-219, August, 1984

[DSV88] M. Dincbas, H. Simonis, P. Van Hentenryck
Solving a Cutting-stock Problem in Constraint Logic Programming.
Fifth International Conference on Logic Programming, Seattle, USA, August, 1988

[DSV92] M. Dincbas, H. Simonis, P. Van Hentenryck
Solving a Cutting-stock Problem with the Constraint Logic Programming Language CHIP. Math. Comput. Modelling, Vol 16, Nr 1, pp 95-105, 1992

[SA97] H. Simonis, A. Aggoun
Search Tree Visualization Tool, COSYTEC Technical Report, November 1997