

# News on Producers and Consumers

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

## 1 Abstract

In this report we describe a variant of the well-known producer/consumer constraint occurring in process-based scheduling problems. For this type of situation the original producer/consumer model with cumulative does not provide satisfactory propagation. We discuss several other constraint models, either based on cumulative or on diffn and show some improvements on a typical example. Adding some redundant implications further improves the reasoning, so that the simple example problem is solved by propagation only. The resulting model with diffn and implications can be useful for a number of other producer/consumer situations, if an initial order of the tasks can be given.

## 2 Introduction

The producer/consumer constraint [SC95] can be used in many scheduling problems, where stock levels must be controlled over a given time period. The original problem description is very general, handling any set of producer and consumer tasks. In particular, no order on the tasks is required. The constraint can be expressed with the cumulative [AB93] constraint of CHIP, which provides good propagation results for many problem instances. Typical applications control the intermediate product stock-levels in (semi-) process scheduling problems, or handle constraints on the availability of raw-materials or storage space for finished products. The constraint is thus used in a number of large-scale CHIP applications and usually works quite well.

In this paper we describe a problem variant for which the constraint propagation with the standard model is not sufficient. We have extracted a small example from a larger scheduling problem to illustrate the difficulty.

We are considering a scheduling problem in the process industry. A single production line is running continuously to produce chemicals of different quality grades. The production is scheduled by switching from one grade to another at given time points. Customer orders are not handled individually, but are grouped together into a general demand, which is typically satisfied from stock. The production schedule must replenish the stock so that each material is always available. On the other hand, finished product storage is quite limited and is dedicated to the different product grades. The scheduling problem consists in defining start and duration of production runs for each product quality over a one-month period. Set-up constraints limit the possible sequences of products due to clean-out times and the wish to limit product down-grading. For each product grade, we can consider a producer consumer problem. An unknown number of production runs of unknown duration plus a given, initial stock must satisfy the demand at each time point, while never exceeding the limited storage capacity. A typical solution consists in making production runs as late

as possible for each product, to avoid problems with the storage capacity. At the same time, the run length is set either to a typical value (often process dependent) or to the maximal length that the storage capacity allows. The need to co-ordinate the runs for different products, considering setup and individual demand, creates the challenge in this scheduling problem.

### 3 Problem Description

The problem described above is too complex for a detailed study. In the following we concentrate on the following sub problem. We consider a sequence of production runs for one product only. The product is made on a single, disjunctive machine, so that an order of the production runs can be given. We fix the number of runs a priori to a large number (possible runs). Only those runs before a cut-off date will be actually made, the others consist of dummy runs. A production run can last between 1 and 2880 minutes, with a preferred time of 1080 minutes. All quantities have been converted into production time, a production run of  $A$  minutes produces  $A$  units of product, which is available at the end of the run. The demand is given for fixed time points, all quantities are also expressed in production time requirements. We ignore the maximal stock level constraint for the moment, and only look at the minimal stock level constraint.

We use the following notation:

$N$	the number of producer events
$M$	the number of consumer events
$L$	the initial stock level
$T_i$	the end time of producer task $i$
$P_i$	the quantity produced in producer task $i$
$S_j$	the time point of consumer task $j$
$C_j$	the quantity consumed in consumer task $j$

We assume that producer and consumer tasks are ordered, i.e.

$$T_i < T_{i+1}$$

$$S_j < S_{j+1}$$

We must produce at least the quantity that we consume, but can produce more if we want to, so the following inequality holds:

$$\sum_{i=1}^N P_i + L \geq \sum_{j=1}^M C_j \quad (1)$$

At each time point, the accumulated production plus the initial stock must exceed the consumption up to this time point. We can express this condition with sets of disjunction between producer tasks and consumer tasks:

$$T_i > S_j \vee \sum_{k=1}^{i-1} P_k + L \geq \sum_{l=1}^j C_l \quad \text{for all } i \text{ in } \{1..N\} \text{ and all } j \text{ in } \{1..M\} \quad (2)$$

This condition states that the total production before consumer task  $j$  (which consists of all producer tasks  $1..i-1$  plus the initial stock) must exceed the total consumption (which consists of all consumer tasks  $1..j$ ).

We can further tighten the inequality constraints between the producer tasks by considering the temporal relation between the ends. If task  $i$  produces  $P_i$  units and finishes at time  $T_i$ , then its start time must be  $T_i - P_i$ . As the production is disjunctive, the previous task must have ended before that time point. We can thus add the constraints

$$T_{i+1} \geq T_i + P_{i+1} \quad \text{for all } i \text{ in } \{1, N-1\}$$

Note that this formula is not the usual time relation between tasks, expressed between start times, but a relation between the end times of the tasks.

For the particular example studied here, the demand is given in the following table:

Time point	Quantity
2880	1440
7200	1440
12960	1440

**Table 1: Data of the small example problem**

The initial stock is given as 720 units, the planning horizon is 28800 units. Any task later than 20000 is considered a dummy run.

We are looking for a solution satisfying two criteria:

- Each run should be as late as possible
- Each run should have the preferred quantity 1080, except for dummy runs.

We consider as objective A the situation where preference is given to the lateness of the runs, and objective B where preference is given to the run size. Both problems have different solutions, as we will show below.

## 4 Constraint Models

We now describe the different constraint models and their CHIP programs. The programs are intended for the demonstration of some technique, nearly all can be made more efficient by adding accumulators, etc. If a predicate was already shown in an earlier program version, it is not listed again. The section names (producer?) indicate name of the program in the problem directory.

### 4.1 producer

This program uses the standard expression of the producer/consumer constraint with cumulative [SC95].

```
top(Producer):-
    consumers(Consumer),
    producers(Producer),
    Initial_stock = 720,
    Final_stock :: 0..2880,
    End = 28800,
    producer_consumer(Producer,Consumer,Initial_stock,Final_stock,End),
    fix_values(Producer,1).
```

```

consumers([cons(2880,1440),cons(7200,1440),cons(12960,1440)]).

producers([prod(T1,Q1),prod(T2,Q2),prod(T3,Q3),prod(T4,Q4),prod(T5,Q5),
  prod(T6,Q6),prod(T7,Q7),prod(T8,Q8),prod(T9,Q9),prod(T10,Q10)):-
  [T1,T2,T3,T4,T5,T6,T7,T8,T9,T10] :: 0..28800,
  [Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10] :: 1..2880,
  T1 #>= Q1,
  T2 #>= T1+Q2,
  T3 #>= T2+Q3,
  T4 #>= T3+Q4,
  T5 #>= T4+Q5,
  T6 #>= T5+Q6,
  T7 #>= T6+Q7,
  T8 #>= T7+Q8,
  T9 #>= T8+Q9,
  T10 #>= T9+Q10.

producer_consumer(Producer,Consumer,Initial_stock,Final_stock,Final):-
  Limit :: 0..32000,
  producer_tasks(Producer,Start1,Dur1,Res1,End1,Initial_stock,Sum),
  consumer_tasks(Consumer,Final,Start2,Dur2,Res2,End2),
  append(Start1,Start2,Start),
  append(Dur1,Dur2,Dur),
  append(Res1,Res2,Res),
  append(End1,End2,End),
  Limit #<= Sum,
  cumulative(Start,Dur,Res,End,unused,Limit,Final,unused).

producer_tasks([],[],[],[],[],S,S).
producer_tasks([prod(T,Q)|P1],[0|S1],[T|D1],[Q|R1],[T|E1],Sum,Send):-
  producer_tasks(P1,S1,D1,R1,E1,Q+Sum,Send).

consumer_tasks([],_,[],[],[],[]).
consumer_tasks([cons(T,Q)|C1],Final,[T|S1],[D|D1],[Q|R1],[Final|E1]):-
  D :: 0..30000,
  T + D #= Final,
  consumer_tasks(C1,Final,S1,D1,R1,E1).

fix_values([],N).
fix_values([prod(T,Q)|R],N):-
  indomain(T,max),
  indomain(Q,1080),
  N1 is N+1,
  fix_values(R,N1).

```

### Program 1: The basic program

The producer tasks have variable height, and the labeling routine tries to set this height to a value starting from 1080, after scheduling the task as late as possible. This program does not work at all! It start enumerating the first task from the time point 28791, the producer/consumer constraint does not restrict the domain. This is due to three main factors:

- The number of producer tasks needed is unknown, we use the dummy tasks to handle this uncertainty.
- The quantity in the producer tasks is not known, neither their time point. Any cumulative reasoning does not have enough information to deduce required times or quantities.
- The problem is even more complex since the total resource limit in the constraint is also not known, but only constrained to be the sum of the initial

stock and all producer tasks. Given the wide range of possible values for the resource limit, useful propagation is not possible.

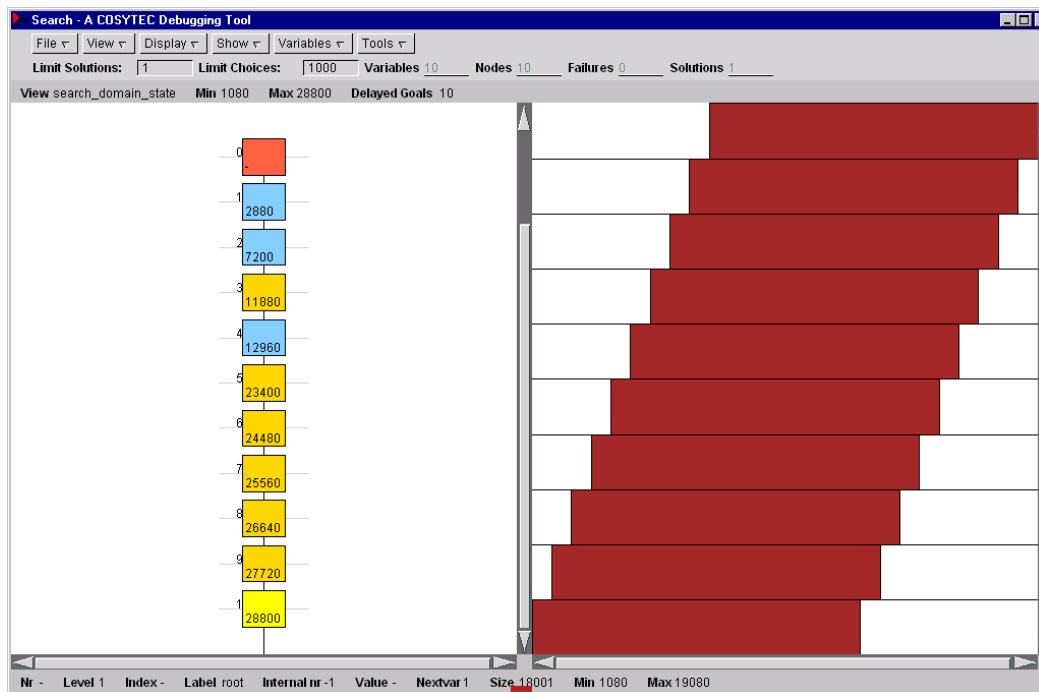
#### 4.2 *producer1*

As a first modification, we can restrict the problem by enforcing the size of the producer tasks. If we want the tasks to have size 1080, we should perhaps force this size already in the domain definition. This will also fix the overall resource limit to an integer value. We change one line in the program:

```
[Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10] :: 1080..1080,
```

#### Program 2: Fixing the producer size

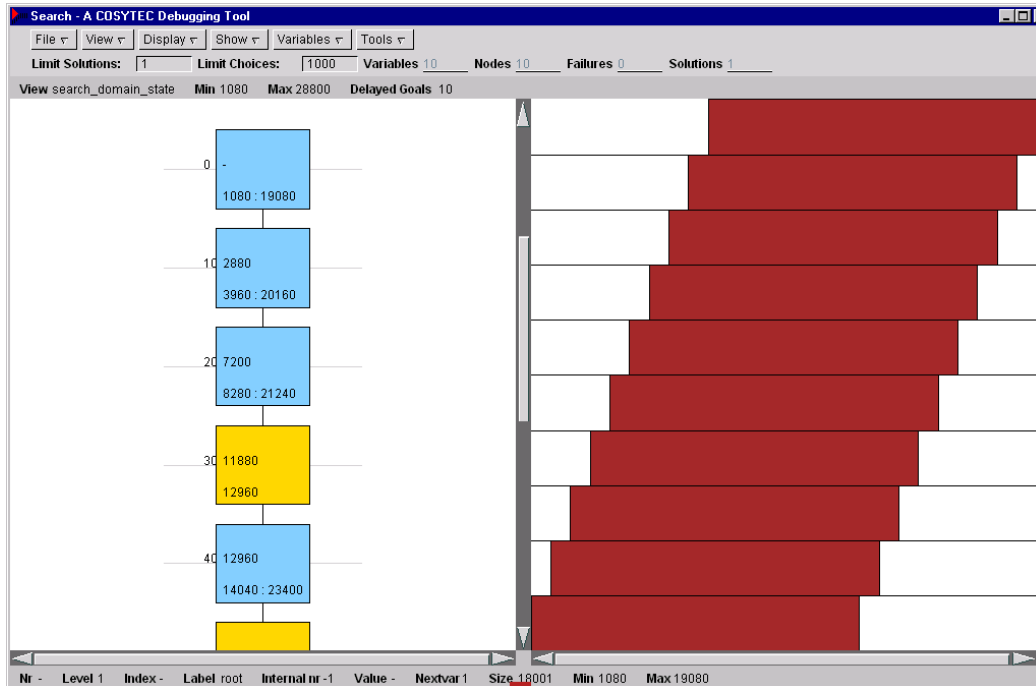
We now obtain a solution, instrumentation indicates zero backtracking steps and the search tree [SA97] generated by the program (shown below in figure 1) shows a straight line!



**Figure 1: The ideal search tree?**

In each node on the left, we see the selected value for the variable. On the right we see the initial domain of all variables before starting the search, the root-node is selected in the tree.

All is well, isn't it? Not quite, since the program still runs for a rather long time. We can see what is happening by selecting the option 'Show domains' in the search tree tool (figure 2):



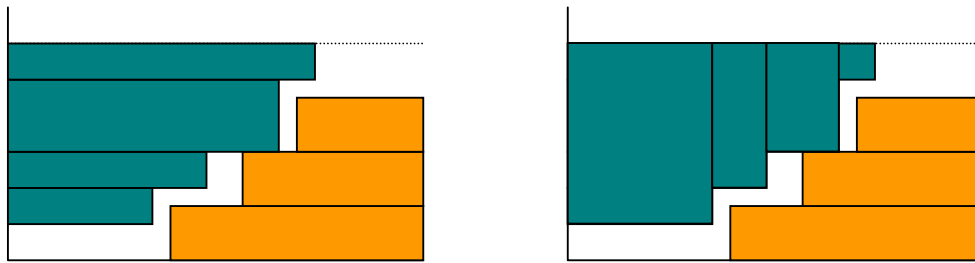
**Figure 2: Problems with propagation**

For the first variable, the domain is 1080..19080, but the selected value is 2880. Remember that we use  $\text{indomain}(X, \max)$  to assign values. This means that we have tested all values from 19080 downwards, before finding the consistent value 2880. Obviously, we spend a lot of time exploring these immediate failures. The constraint propagation is too weak and does not remove all inconsistent values.

It is important to note that this type of problem occurs with many constraint programs, but is rarely explained/mentioned in the result analysis. The classical backtracking count does not recognize this, neither does a simple search tree. Only looking at the domain values before and after the assignment we can find this cause of problems.

### 4.3 *producer6*

How can we improve these rather disappointing results? The first attempt is to re-organize the cumulative constraint to improve the reasoning. In the original model, the producer tasks are created as long, flat rectangles. A restriction on one producer is only visible when we know the height of the other producers and their impact on the overall resource limit. With variable height of tasks, this impact is very weak and the resulting propagation not satisfactory. But we can re-organize the producer tasks in this problem, since we know that they are sorted in time. Instead of using wide, flat rectangles, we can use narrow, high rectangles as shown in figure 3 below:



**Figure 3: Horizontal or vertical producer tasks**

Obviously, the area occupied by the producer tasks is the same, both formulations are equivalent. But given say the first task, we no longer need to know all other tasks to do some deduction.

Unfortunately, this improvement is not enough to handle the case of variable height tasks. Even with this modification, the program still continues to backtrack early. If we fix the task height to 1080 by domain definition, the modified constraint reduces the domains a bit more, but still not sufficiently. Table 2 compares the propagation of programs producer1 and producer6. It shows the domain of each variable just before its assignment and the first consistent value found. The highlighted fields show when the propagation has removed all inconsistent values above the first consistent one.

producer1	producer6	first value chosen
1080..19080	1080.. <b>2880</b>	2880
3960..20160	3960..9000	7220
8280..21240	8280.. <b>11880</b>	11880
<b>12960</b>	12960..21600	12960
14040.. <b>23400</b>	14040.. <b>23400</b>	23400
<b>24480</b>	<b>24480</b>	24480
<b>25560</b>	<b>25560</b>	25560
<b>26640</b>	<b>26640</b>	26640
<b>27720</b>	<b>27720</b>	27720
<b>28800</b>	<b>28800</b>	28800

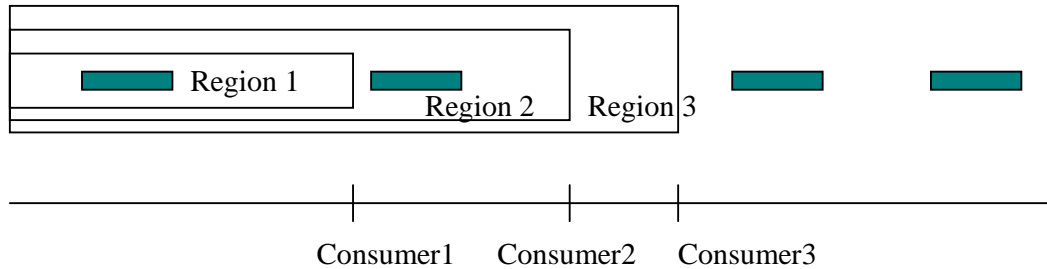
**Table 2: Propagation results**

**4.4 producer2**

While the program above shows some improvements for the fixed size case, it does not handle the variable height situation at all. What alternatives do we have?

Scanning through the CHIP reference manual, we find a possible candidate in the region parameter of the diffn [BC94] constraint. The region parameter checks the utilization by tasks for a given region in an n-dimensional space. Different variants are possible, counting the difference between first and last utilization, the overall space used or the overall amount used in one dimension only. The parameter is normally used to restrict the amount of work that can be placed into a certain area, for example how much work can be performed by one machine in a given time period. We can (mis-)use this parameter to express that in the time before a consumer demand

we want to producer at least the total consumption up to this point minus any initial stock. For each consumer task, we generate a new region with a new utilization domain variable, for which we constrain the minimum of the value. Each producer task is represented with a task of height one, duration equal to the quantity produced and start and end times according to the scheduled time. Figure 4 shows the layout:



**Figure 4: Diffn constraint with region constraints**

Program 3 shows the code of this program.

```

top(Producer):-
    consumers(Consumer),
    producers(Producer),
    Initial_stock = 720,
    Final_stock :: 0..2880,
    End = 28800,
    producer_region(Producer,Consumer,Initial_stock,Final_stock,End),
    fix_values(Producer,1).

producer_region(Producer,Consumer,Initial_stock,Final_stock,Final):-
    producer_region_tasks(Producer,Rectangles),
    producer_region_regions(Consumer,Initial_stock,Regions),
    diffn(Rectangles,unused,unused,unused,unused,Regions).

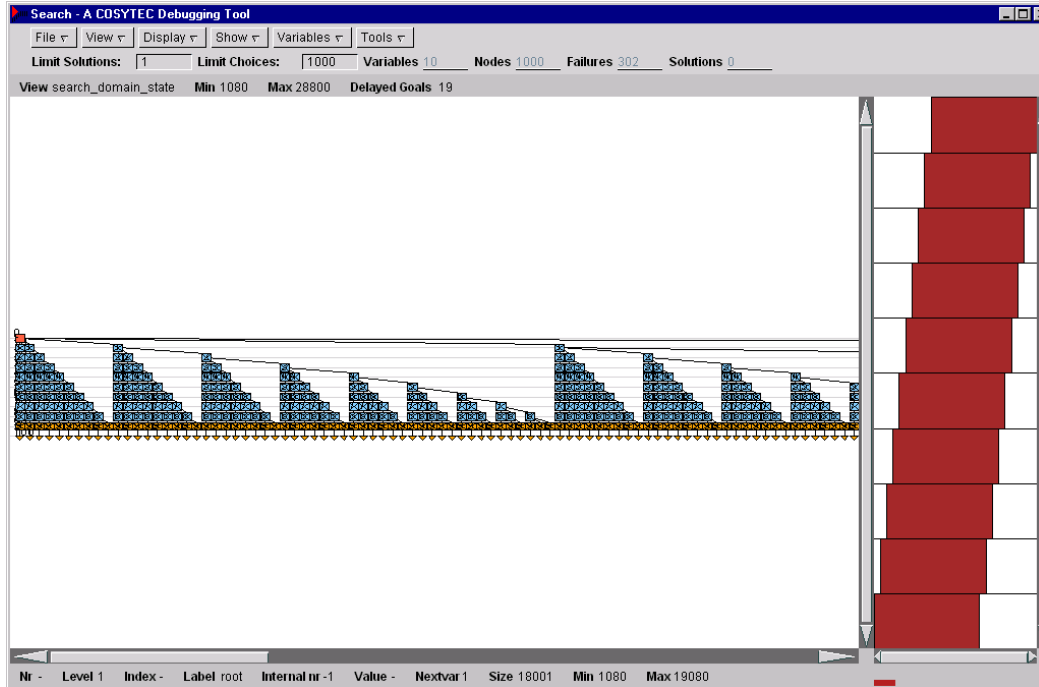
producer_region_tasks([],[]).
producer_region_tasks([prod(T,Q)|P1],[[X,Y,W,H]|R1):-
    X :: 0..32000,
    T #= X+Q,
    Y = 0,
    W = Q,
    H = 1,
    producer_region_tasks(P1,R1).

producer_region_regions([],_,[]).
producer_region_regions([cons(T,Q)|C1],Stock,[[[0,0,T,1],-1,Use]|R1):-
    Q1 is max(0,Q-Stock),
    Use :: Q1..T,
    producer_region_regions(C1,Stock,R1).

```

### Program 3: Using regions in diffn

How does this variant measure up? Figure 5 shows the generated search tree:



**Figure 5: Search tree for region constraint**

This is a complete and total failure, the constraint only updates the lower limit of the utilization once all tasks have been assigned. This indicates a missing propagation method in the diffn region constraint. The form of the search tree is typical for a very passive constraint. It always fails at the same level, the complete tree above the failure is generated and it does not find a solution within a reasonable time limit.

Well, perhaps in the next version of CHIP we will have the proper propagation rule...

The model itself is actually quite nice, as it allows to partially allocate some production to a consumer, if the producer task lies partially in the corresponding region.

Note that we can not invert the regions and state limits on the maximum utilization of time after the consumer event, as we do not know the total quantity that will be produced.

#### 4.5 *producer3*

What next? Well we can use the diffn constraint for other things as well. In the original producer/consumer model we took figure 3 and treated it as a cumulative problem. But we can also see this as a 2D placement problem! The consumer tasks are fixed, so they correspond to fixed rectangles. The producer tasks must be stacked on top of each other. We can do this easily by linking the y-coordinates of the tasks with equality constraints. We again use here the fact that the temporal sequence of the tasks is given. In the general producer/consumer situation we could not express this rules so easily. Program 4 shows the program code:

```
top(Producer):-
    consumers(Consumer),
    producers(Producer),
    Initial_stock = 720,
    Final_stock :: 0..2880,
    End = 28800,
```

```

producer_placement(Producer,Consumer,Initial_stock,Final_stock,End),
fix_values(Producer,1).

producer_placement(Producer,Consumer,Initial_stock,Final_stock,Final):-
  producer_rect(Producer,Initial_stock,Rect1),
  consumer_rect(Consumer,Final,0,Rect2),
  append(Rect1,Rect2,Rect),
  diffn(Rect).

consumer_rect([],_,_,[]).
consumer_rect([cons(T,Q)|C1],Final,Y,[[T,Y,W,Q]|R1]):-
  Y1 is Y+Q,
  W is Final-T,
  consumer_rect(C1,Final,Y1,R1).

producer_rect([],_,[]).
producer_rect([prod(T,Q)|P1],Level,[[0,Y,T,Q]|R1]):-
  Y :: 0..30000,
  Y #= Level,
  producer_rect(P1,Y+Q,R1).

```

#### Program 4: Producer/Consumer as 2D placement

The resulting search tree for the fixed height situation is shown below in figure 6:

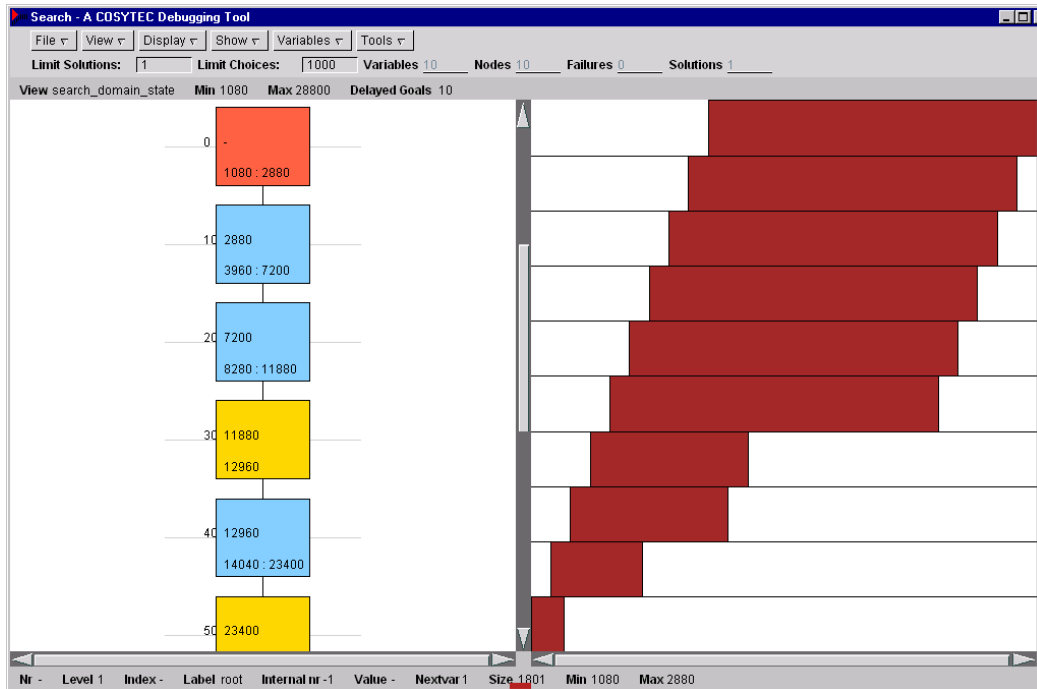


Figure 6: Search tree for placement variant

The tree on the left shows that the domain restrictions are now much better than before (compare to table1), the initial domains on the right also show this improvement. In fact, for the case of fixed height this is the best possible result.

#### 4.6 producer4

If we remove the restriction on the producer task size, then the program is not yet ideal. We can distinguish two situations.

#### 4.6.1 Objective A

For each producer task, we assign first the start time (as late as possible) and then decide on the quantity. This finds a solution without backtracking, where task3 has size 1440, tasks 1 and 2 have size 1080.

#### 4.6.2 Objective B

If we assign the quantity first and then fix the start, we still need a very long backtracking before finding the solution. For task3, after assigning size 1080, a wrong time is chosen (too late). There is not enough time before the third consumer task to make the missing quantity. Again, we have a lack of propagation here which forces us to enumerate many inconsistent values before finding the solution with a task4 of size 360.

#### 4.7 producer5

Can we improve our placement model? One possibility is to use distance constraints between the producer tasks in addition to the equality constraints between the y-coordinates. The distance between consecutive producer tasks is zero, so that this constraint can be stated very easily. But for this problem, it does not improve the propagation.

#### 4.8 producer7

To further improve the program, we have to go back to its definition in formula (2). We can derive from this formula the following implication, which is a necessary condition for (2)

$$T_i > S_j \Rightarrow \sum_{k=1}^{i-1} P_k + L \geq \sum_{l=1}^j C_l \quad \text{for all } i \text{ in } \{1..N\} \text{ and all } j \text{ in } \{1..M\} \quad (3)$$

In this formula, we can replace the time needed to produce task i with the time that is left after producing task i-1, but before the consumer task j. The following implication still holds:

$$T_i < S_j \Rightarrow \sum_{k=1}^i P_k + L + S_j - T_i \geq \sum_{l=1}^j C_l \quad \text{for all } i \text{ in } \{1..N\} \text{ and all } j \text{ in } \{1..M\} \quad (4)$$

In CHIP, we can add this implication with conditional propagation in the form of a if-then-else constraint. Program 5 below shows the CHIP code:

```
top(Producer):-
    consumers(Consumer),
    producers(Producer),
    Initial_stock = 720,
    Final_stock :: 0..2880,
    End = 28800,
    producer_placement(Producer,Consumer,Initial_stock,Final_stock,End),
    producer_generate(Producer,Consumer,Initial_stock,0),
    fix_values(Producer,1).

producer_generate(P,[],_,_).
producer_generate(P,[cons(S,C)|C1],L,Made):-
    producer_generate1(P,S,C,L,Made+C),
```

```

producer_generate(P,C1,L,Made+C).

producer_generate1([],S,C,L,Made).
producer_generate1([prod(T,Q)|P],S,C,L,Made):-
    if T #> S then L #>= Made,
    if T #< S then L+Q+S #>= Made+T,
    producer_generate1(P,S,C,L+Q,Made).

```

### Program 5: Additional Implications

This addition solves the problem. We find solutions for both objectives A and B without backtracking, just by propagation without testing inconsistent values.

#### 4.9 *producer8*

The last modification is to remove the *diffn* constraint and to keep the implications only. This lands us back at square one. There is not enough propagation, we start enumerating inconsistent values. As we have replaced disjunction (2) with an implication, this is not surprising. We only get the propagation in one direction. In order to solve the complete problem by inequalities, we would have to introduce additional 0/1 variables to model the disjunction.

## 5 Conclusions

We have considered a number of alternative models for the producer/consumer constraint where tasks are ordered, but number and size of the tasks are not determined a priori. What lessons we can learn from this experience?

### 5.1 *Standard model not sufficient*

The standard model using cumulative was not sufficient to solve even a simple example. The uncertainty about the domain variables is too big to perform good propagation. Changing the model from horizontal to vertical producer tasks improves the propagation, but not enough.

### 5.2 *Region model not useful*

The region constraint in *diffn* does not provide constraint propagation in the required direction. It is intended to restrict the maximum usage in a region, not to enforce a minimum usage as required here. The trick of inverting the region and constraining everything outside it does not work as the total usage is not known a priori.

### 5.3 *Placement model ok, but not perfect*

The 2D placement model of the producer/consumer constraint requires the fact that the tasks are ordered in time. It performs much better than the standard model, but not well enough. If we add some necessary implications, we finally obtain very good propagation.

### 5.4 *No backtracking is not enough*

As we could see on program *producer1*, it is not enough to have zero backtracks in a solution. We must also consider the effort required to remove inconsistent values, which immediately fail after the assignment. The search tree tool can be used to discover and understand such problems.

### **5.5 Use of search tree tool**

The search tree tool [SA97] and the global constraint visualization [ABB+98] can provide useful hints to understand what is happening in a program. They also provide a simple way to explain problems to others without writing expensive ad-hoc programs.

### **5.6 Think small**

It is always a good idea to isolate a small part of a larger application problem in order to understand the basic propagation mechanism. It would be quite difficult to directly understand the producer/consumer constraint behavior within the context of a large scale application with multiple production lines, dozens of products and all kinds of other constraints. Once the basic constraint behavior is understood, we can much more easily understand what is happening in the larger application.

## **6 Bibliography**

[AB93] A. Aggoun, N. Beldiceanu

Extending CHIP in Order to Solve Complex Scheduling Problems

Journal of Mathematical and Computer Modelling, Vol. 17, No. 7, pages 57-73

Pergamon Press, 1993

[ABB+98] A. Aggoun, N. Beldiceanu, E. Bourreau, H. Simonis

Visualization of Global Constraints

COSYTEC Technical Report, October 1998

[BC 94] N. Beldiceanu, E. Contejean

Introducing Global Constraints in CHIP

Journal of Mathematical and Computer Modelling, Vol 20, No 12, pp 97-123, 1994

[SC95] H. Simonis, T. Cornelissens

Modelling Producer/Consumer Constraints

Proc. Principles and Practice of Constraint Programming, Cassis, France, September 1995

[SA97] H. Simonis, A. Aggoun

Search Tree Debugging

Technical Report, COSYTEC SA, 1997