# THE SEQUENCE PUZZLE

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***N*** | ***Base*** | | ***Base + Implied*** | | ***Global*** | | ***Global + Implied*** | |
|  | *Fails* | *Time* | *Fails* | *Time* | *Fails* | *Time* | *Fails* | *Time* |
| **500** | 617 | 37s 461msec | 495 | 23s 734msec | 989 | 1s 5msec | 493 | 565msec |
| **1000** | 1,247 | 4m 14s | 995 | 1m 44s | 1,989 | 2s 222msec | 993 | 1s 28msec |

*Going from Base → Global, and going from Base + Implied → Global + Implied: what is the main advantage of using a global constraint? Why?*

Moving from the base model to the global model we see that the main advantage of using a global constraint is in the more efficient propagation of the global constraints in the global model.

The basic model has fewer failures, but the use of global constraints allows the problem to be solved more efficiently and requires less time. This happens because global constraints know the correlations between the various elements of the problem and provides a more complete view of the problem than decomposed constraints.

*Is there an implied constraint that now becomes redundant in the Global + Implied model? Why?*

The GCC can completely replace the first implicit constraint, simplifying the model without losing its effectiveness in solving the problem, because the GCC provides all the information already present in the 1st implicit constraint.

So, in the "Global + Implicit" model the 1st implicit constraint is redundant because it is sufficient to use the globalCardinality (GCC) constraint.

Well Done! As for your comments:

nQueens

I think you need to unify your explanations in the last two paragraphs. In the last one, you talk about “fewer failures” but as a justification you are referring to the modelling benefits and efficiency. Instead, the reason is what you said in the previous paragraph.

Poster

Similarly, you seem to refer to propagation strength & efficiency interchangeably. It would be nice if you separate the two benefits of global constraints (both in nQueens and Poster) and write your explanations accordingly: propagation strength and propagation efficiency.

Puzzle

Yes, sometimes the global constraints are useful mainly due their efficiency in propagation rather than in the amount of propagation as seen in this example. It would be nice if you can explain better where this efficiency comes from (related to my comment above). The sentence “the correlations between the various elements of the problem “ is rather vague.

In any case, why do you think the base models (even the base+implied) are too costly timewise?

Can you explain why the gcc provides the information present in the first implied constraint?

You can write here in the comments without a new submission.

N-QUEENS  
To clarify what we have written in the last two paragraphs, constraint decomposition does not always provide efficient propagation, in fact it is a very effective and efficient method only for some global constraints, but cannot be used in all cases. These results are due to the fact that the Generalized Arc Consistency (GAC) on the original global constraint is often stronger than the (G)AC on the decomposition constraints.  
  
POSTER  
For both nQueens and Posters, we explain the concepts of “propagation strength” and “propagation efficiency” related to these two problems.  
By “propagation efficiency” we mean that global constraints allow a more efficient and faster resolution of the problem by exploiting the structure of the problem.  
By ‘propagation strength’, on the other hand, we mean that global constraints capture the relationships between the variables, ensures that the constraints are respected and this reduces the possibility of failures in the model.  
  
PUZZLE  
The efficiency comes from the fact that the implicit constraints used are able to give more information and allow for better propagation efficiency.  
Regarding time, we did not mean that they are “too expensive” but only that they require more time to arrive at the solution for the same reason as stated in the previous sentence. These constraints do not fully exploit the structure and relationships of the problem, resulting in longer solution times.  
Gcc provides the information in the first implicit constraint because it says that the sum of all elements in the array x must equal n. In fact, the first implicit constraint states just that (that the sum must equal n).

Global constraints  
  
With strength, we mean stronger resolution which is related to what you said above for nQueens (which is not very clear instead in your Poster answer). For instance, if the propagation algorithm maintains GAC, it will detect many more inconsistent values compared to a decomposition. In the poster, you say "ensures that the constraints are respected" but this is the case anyway with the decomposition, isn't it?  
  
With efficiency, yes we mean faster resolution, but how do they do it? Exploiting the problem structure may not always give an efficient algorithm. We discussed some examples in the class.  
  
Puzzle  
  
Base model's propagation strength is very similar to the GCC's if you look at the failures. Instead the base model is too costly in time. I would like that you understand this. HINT: Look at the solution statistics, you will see that there are many variables and constraints in the base model. Try to understand where they come from, and quantify them in terms of n. The answer will tell you why the model is costly.  
  
You say "Gcc provides the information in the first implicit constraint because it says that the sum of all elements in the array x must equal n." which is not true. GCC constraint ensures sth else. Please study better the relationship between GCC and the sum constraint.

Global constraints  
  
COMMENT: With strength, we mean stronger resolution which is related to what you said above for nQueens (which is not very clear instead in your Poster answer). For instance, if the propagation algorithm maintains GAC, it will detect many more inconsistent values compared to a decomposition. In the poster, you say "ensures that the constraints are respected" but this is the case anyway with the decomposition, isn't it?  
OUR ANSWER: Yes, the constraints are still respected, but you can see that with decomposition (dividing the problem into many smaller sub-constraints) the time increases a lot when the problem size increases and there is a much higher computational complexity.  
  
COMMENT: With efficiency, yes we mean faster resolution, but how do they do it? Exploiting the problem structure may not always give an efficient algorithm. We discussed some examples in the class.  
OUR ANSWER: They achieve faster resolution because, in addition to exploiting the structure of the problem, they use specialised propagation and specific global constraints. Specific global constraints allow for a much smaller number of solutions to be analysed, so that the best solutions can be found in less time.  
  
  
Puzzle  
  
COMMENT: Base model's propagation strength is very similar to the GCC's if you look at the failures. Instead the base model is too costly in time. I would like that you understand this. HINT: Look at the solution statistics, you will see that there are many variables and constraints in the base model. Try to understand where they come from, and quantify them in terms of n. The answer will tell you why the model is costly.  
OUR ANSWER: We reviewed the statistics and actually noticed that a really high number of Boolean variables (and even integer variables) are created with the basic model (compared to the global model which generates fewer). For these reasons, the basic model takes much longer as most of it is spent on compiling and processing.  
Another difference we have noticed is that propagation is much higher in the basic model: the global model manages to process the variables involved in a unique way, whereas the basic model requires more propagation due to the large number of constraints it needs.  
  
COMMENT: You say "Gcc provides the information in the first implicit constraint because it says that the sum of all elements in the array x must equal n." which is not true. GCC constraint ensures sth else. Please study better the relationship between GCC and the sum constraint.  
OUR ANSWER: We tried to go deeper and realised that the first implicit constraint is redundant because it does not improve propagation because it contains information that is already contained in the global cardinality constraint (in the third parameter the occurrences are counted).  
We are able to have the cardinality of vj because each Oj allows us to calculate how many times vj is present in the sequence and, consequently, we have the cardinality of Xj because it is assigned to vj.  
The GC constraint can know all the information provided by the first implicit constraint because the GC constraint collects the count and ensures that the count is bounded to the number of occurrences of vj.

We better talk about these topics together. Please approach to me at the next lecture.

BECAUSE THE BASIC MODEL IS TOO EXPENSIVE IN TERMS OF TIME?  
  
With the GCC we have n variables and 1 constraint.  
  
The basic model, on the other hand, works in a different way.  
  
The sum decomposition works this way:  
– X1+X2=Y1  
– Y1+X3=Y2  
– ...  
– Y(n-1)+Xn = N  
  
The sum, then, gives us 1 variable per constraint, so n constraints.  
We have refied constraints on boolean variables of the form xij = bij, so there are n^2 variables in total.  
We have n constraints for each i from 0..n-1 of the form xi = Bi1+Bi2+....+Bin, which gives us n^2 variables.  
  
In total, therefore, we have 2n^2+n variables which in terms of time result in O(n^2).  
  
For this reason, the basic model takes much longer.  
  
  
  
  
WHY IS THE 1ST IMPLICIT CONSTRAINT REDUNDANT?  
  
The 1st implicit constraint counts the occurrences and checks that the total is equal to n.  
Using the 1st implicit constraint improves the basic model, this is because decomposition does not help as it does not have the entire view of the sequence as it does with the global constraint.  
  
The global\_cardinality constraint allows all values to be considered simultaneously.

The GCC is a count constraint and limits the number of times each value is taken by variables.

In general:

global\_cardinality(array [$X] of var int: x,  
array [$Y] of int: cover,  
array [$Y] of var int: counts)

Requires that the number of occurrences of cover [ i ] in x is counts [ i ].

In our case, it bounds the number of times x takes the value vj to the count of Oj. For this reason, it can take the counts and constrain them to the number of occurrences of vj.

constraint global\_cardinality(x,0..n-1,[x[i] | i in 0..n-1]);

The third parameter of gcc indicates that all summed occurrences are equal to the length of x.  
SUM Oj = length(x)  
The implicit constraint, on the other hand, is constructed as follows: SUM xj = n  
Given that length(x) = n and that O = X, then the constraints are identical.  
  
  
For this reason, we can use GCC instead of the 1st implicit constraint.

------------------------------------------------

“ In the poster, you say "ensures that the constraints are respected" but this is the case anyway with the decomposition, isn't it?”  
  
Yes exactly.  
  
“With efficiency, yes we mean faster resolution, but how do they do it?”  
  
The global model is more efficient because it incorporates specific global constraints and, in addition, uses specialised propagation.  
Global constraints reduce the number of partial solutions to be examined during the search (thus reducing the search space) and facilitate the identification of optimal solutions.  
Another improvement from an efficiency point of view is that the global model utilises incremental computation, which avoids having to compile everything every time it is called.  
  
  
“WHY IS THE BASIC MODEL TOO EXPENSIVE IN TERMS OF TIME?”  
  
Thanks to the (forall i,j x[j]=i <=> b[i,j]==1), we have n^2 given Boolean variables and n^2 refined constraints are created.

In the end, therefore, we have n variables X[1] ... X[n] in the following form:

- X[1] = B[1,1] + B[1,2] + ... +B[1,n]

- ...

- X[n] = B[n,1] + B[n,2] + ... +B[n,n]

and for each of the n terms of the sums we have n sums giving us n^2 again .

For each term (there are n terms) of the sums, there are n sums that give us n^2 again.  
In the end, there are n more variables from the sum decomposition.

“WHY IS THE FIRST IMPLICIT CONSTRAINT REDUNDANT?”  
  
The GCC constraint is expressed like this: global\_cardinality(array [$X] of var int: X, array [$Y] of int: cover, array [$Y] of var int: counts).

It is written such that the number of occurrences of cover [ i ] in X is counts [ i ].  
Let's compare the GCC with the first implicit constraint.

In GCC it is required that the sum of the elements of X[i] over all indices i is equal to n. This is exactly the same claim made in the first implicit constraint, which states that the sum of the occurrences of i in the array x equals i.

The first implicit constraint does not improve propagation because it provides information that the global cardinality constraint already possesses, in fact it counts the occurrences thanks to the third parameter.  
GCC takes counts into account and limits the number of counts equal to the number of occurrences of vj; for this reason, it can already know all the information of the first implicit constraint.  
The explanation is as follows: each Oj counts how many times vj appears in the sequence; this way, it gives the cardinality of Xi (where Xi is assigned to vj).

For these reasons and for the same reasons given in previous answers, the first implicit constraint is redundant.  
  
We hope we have resolved the doubts of our previous answers!

You introduce the GCC constraint as the number of occurrences of cover [ i ] in X is counts [ i ]. So from here how do you derive that "In GCC it is required that the sum of the elements of X[i] over all indices i is equal to n. "? You never explain this (in fact, you had said the same thing in your previous explanation and I had given you the same comment).

\* correction: not refined, but reified constraints.

- We have defined GCC as follows: constraint global\_cardinality(x,0..n-1,[x[i] | i in 0..n-1]);

This constraint states that for every possible value in the range 0..n-1, the number of occurrences of that value in the array x must equal x[i]. In other words, x[i] represents the number of occurrences of value i in array x.

- Once GCC is defined, we explain the sum of occurrences: each x[i] represents the number of occurrences of the value i, so the sum of all x[i] represents the total sum of the occurrences of all possible values in the x array.

- Now let us examine the total sum and the value n: We have specified the constraint n = sum(i in 0..n-1)(x[i]), so we state that the total sum of the occurrences must equal n. This means that the sum of the elements of x[i] over all the indices i must equal n.

To conclude, the fact that the GCC requires that the sum of the elements of X[i] over all indices i be equal to n results from the combination of the GCC constraint that ensures the correct representation of the occurrences of each value in the array x and the constraint that requires that the total sum of these occurrences be equal to n. The GCC ensures that the occurrences are correctly represented, while the second constraint requires that their total sum be specifically equal to n.  
  
Hopefully, we have managed to explain all the steps.

Sorry but you are starting with the sum constraint, instead what you need to show is that, that sum constraint is already ensured by the gcc constraint and therefore we don't need it. I cannot see that in your argument.

Please show that global\_cardinality(x,0..n-1,[x[i] | i in 0..n-1]) ensures that n = sum(i in 0..n-1)(x[i])

We want to write in this comment all the notes taken to avoid making further explanation errors.

NOTES:

gcc([X1,...,Xk],[v1,...,vm],[O1,...,Om]) iff forall j∈{1,...,m} Oj=|{Xi|Xi=vj, 1<=i<=k}|

Oj give us the cardinality of Xi, where Xi is assigned to vj

gcc(Xj, O, Xo times)

Xj=Σ j∈{1,…n-1} (Xj=i)

Xi —> occurrences of I

gcc(X, 0…n-1, [Xi|i∈0…n-1])

Numbers of occurrences of O in X=Xo

[Xi|i∈0…n-1] —> normally total occurrences <=n, but here we are saying that the occurrences are exactly equals to the value of Xi, therefore summing all the occurrences (i.e. all the values assigned to the Xi) are equal to n.

In conclusion, in our model we have gcc (X, 0..n-1, X) which means every Xi is also the number of occurrences of i in X. Considering that the total occurrences cannot exceed n, this suggests that the sum of Xi equals n.