Decision Making with Constraint Programming

Exercise 2

Dessì Leonardo [0001141270]

Spini Riccardo [0001084256]

Corso di laurea magistrale in Informatica

Alma Mater Studiorum
Università di Bologna

October, 2023

1 | N-Queens problem

Table below reports failures and total time of the Alldifferent model implemented by using global constraints or by decomposing them.

n	Alldifferent GC		Decomposition	
	Fails	Time	Fails	Time
28	78,947	3s 566ms	417,027	9s 168ms
29	31,294	0s 665ms	212,257	2s 196ms
30	1,588,827	23s 468ms	7,472,978	1m 7s

Analysing the results we can see that the presence of global constraints has a strong impact on this specific model for the n-queens problem.
Looking at the results we can see that:

- **Number of Fails**: In the "Alldifferent GC" column, the number of failures is lower. This suggests that the global constraint approach is better at pruning inconsistent solutions, and it often leads to faster convergence.
- **Execution Time**: The "Decomposition" approach takes more time for most cases, indicating that it is computationally more expensive. The "Alldifferent GC" is faster, likely due to specialized propagation techniques associated with global constraints.

The main reasons for such behavior are:
Global constraints often lead to more powerful inference during propagation, which can reduce the search space faster. This is why the "Alldifferent GC" results show fewer failures. Global constraints are optimized for efficient propagation and may have dedicated algorithms that provide strong and efficient propagation.
Decomposition into simpler constraints may not always provide effective propagation. In some cases, a simple combination of sub-constraints does not capture the full power of the global constraint. Moreover, Global constraints provide a means to ensure that all variables in a constraint problem are globally arc consistent (GAC) or bound consistent (BC) simultaneously. In contrast, when decomposing a global constraint, it typically only achieves arc consistency on smaller constraints, usually binary constraints involving pairs of variables.
Using ad-hoc algorithms dedicated to specific global constraints can be more efficient than propagating the individual constraints of the decomposition. Ad-hoc algorithms often exploit incremental computation, meaning they don't recompute everything each

time, resulting in faster execution. Incremental calculation in the context of constraint programming refers to an optimization that avoids having to re-run the solution of a problem from zero when changes are made to the constraints or variables of the problem. In fact:

- At the first call, some partial results are cached.
- At the next call, the cached data is utilised.

Additionally, Global constraints help to model complex problems more concisely. They can express constraints that are difficult to represent using only primitive constraints. This is especially beneficial when the problem naturally has global structures or patterns.

2 | Poster problem

The model that has been implemented is as follows.

- **Parameter**:
  - n: This is an integer parameter representing the number of posters.
  - w: integer parameter representing the width of the sheet of paper.
  - h: integer parameter representing the height of the sheet of paper.
  - ws: array of length n containing the widths of the posters.
  - hs: array of length n containing the heights of the posters.
- **Variables**:
  - $xpos$: array of integer variables, representing the horizontal positions of the posters on the paper.
  - $ypos$: array of integer variables, representing the vertical positions of the posters on the sheet of paper.
- **Domains**:
  - $xpos[i]$: for each poster i takes values between 0 and $w - 1$.
  - $ypos[i]$: for each poster i takes values between 0 and $h - 1$.
- **Constraints**:

- `forall(i in 1..n)(xpos[i] + ws[i] <= w)`: This constraint ensures that the sum of the horizontal coordinates of the poster i (`xpos[i]`) and the width of the poster itself (`ws[i]`) do not exceed the total width of the sheet of paper (w). In other words, it prevents posters from going outside the right-hand edge of the sheet.
- `forall(i in 1..n)(ypos[i] + hs[i] <= h)`: This constraint is similar to the previous one, but affects the vertical coordinates and height of the posters, ensuring that the posters do not go outside the bottom edge of the sheet.
  * We used in **Naïve model**: `forall(i,j in 1..n where i < j)(xpos[i] + ws[i] <= xpos[j] \/ xpos[j] + ws[j]<= xpos[i] \/ ypos[j] >= ypos[i] + hs[i] \/ ypos[j] + hs[j] <= ypos[i])`
    This constraint ensures that posters do not overlap each other.
  * We used in **Global model** `constraint diffn (xpos, ypos, ws, hs)`: This constraint ensures that posters do not overlap each other. It enforces that the regions occupied by posters are disjoint.

Table below reports failures and total time of the Poster Problem implemented by using Naïve model or by Global model.

Instance	Naïve Model		Global Model	
	Fails	**Time**	**Fails**	**Time**
19x19	1,678,013	15s 769ms	300,649	2s 438ms
20x20	2,504,120	23s 709ms	2,030	225ms

Analysing the results we can see that the presence of global constraints has a strong impact on this specific model for the Poster Problem.
The **Naïve model** uses constraint decomposition, meaning that the global problem is decomposed into smaller, simpler sub-constraints, each of which has a known propagation algorithm. In this case, the Naïve model might have decomposed the problem into a large number of smaller sub-constraints. This can lead to high computational complexity and requires a lot of time to find a solution. Furthermore, the computation time increases significantly as the problem size increases (e.g., from 19x19 to 20x20). The Naïve model requires a lot of time to search for a solution due to the complexity of the decomposition and the lack of effective global propagation.
The **Global model** uses specific global constraints that better capture the complex structure of the problem. These global constraints incorporate specialized propagation, which

leverages the problem's structure to reduce the number of searches and the time required to find a solution. The Global model performs much better than the Naïve model in terms of time and the number of failures. Efficient global propagation and maintaining Global Arc Consistency (GAC) significantly reduce the search time. Moreover, the Global model can solve the 20x20 problem in less than a second, while the Naïve model takes several minutes.

The Global model is more efficient and performant because it incorporates specific global constraints and utilizes specialized propagation. These global constraints significantly reduce the number of partial solutions to be examined during the search and make it easier to detect optimal or satisfactory solutions. Additionally, the Global model makes use of incremental computation, meaning it can avoid recomputing everything each time it's called, further enhancing efficiency. In general, global constraints and specialized propagation can make a significant difference in solver performance, especially for complex problems like this one.

The images below show a graphical representation of one solution to the poster problem with the given instance data. We have implemented this representation with a script in *python*. The figure (2.1) represents a solution of 19×19 instance and the figure (2.2) a 20×20 instance.

Note that there are gaps in the figure (2.1) in scattered positions since:

$$\sum_{i=1}^{n} ws_i \cdot hs_i < w \cdot h$$

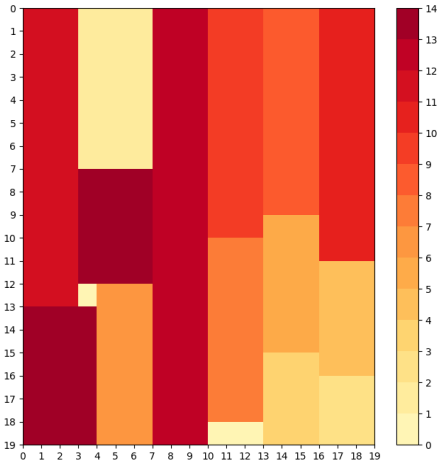

Figure 2.1: 19×19 solution

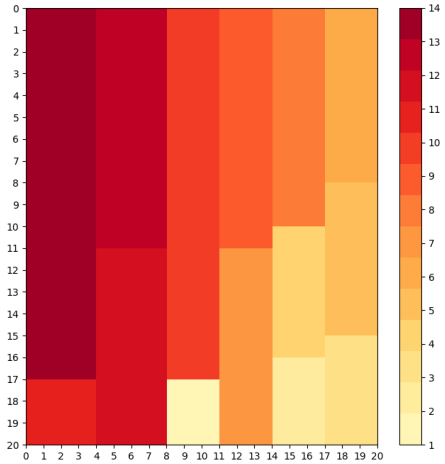

Figure 2.2: 20×20 solution

3 | Sequence problem

Table below reports failures and total time of the Sequence problem implemented by using Base model, Base model with implied constraint, Global model and Global model implied with implied constraint.

n	Base		Base + Implied		Global		Global + Implied	
	Fails	**Time**	**Fails**	**Time**	**Fails**	**Time**	**Fails**	**Time**
500	618	34s	495	25s	989	0s 361ms	493	0s 215ms
1000	1,743	1h 21m	995	1m 38s	1,989	1s 144ms	993	0s 487ms

3.1 | Going from Base → Global

It can immediately be seen that the **Base model** compared to the **Global model** has fewer failures but takes much longer to reach the solution. This happens because much of the time spent in the resolution process in the basic version is spent on compilation. This is due to the fact that the compiler generates hundreds of thousands of Boolean variables to represent the meta-constraint. In fact, as can be seen in the figures 3.1 and 3.2, the compiler with the basic model with $n = 500$ generates hundreds of thousands of auxiliary variables

```
%%%mzn-stat: evaluatedReifiedConstraints=250000
%%%mzn-stat: flatBoolVars=250000
%%%mzn-stat: flatIntConstraints=500500
%%%mzn-stat: flatIntVars=250500
%%%mzn-stat: flatTime=9.63938
%%%mzn-stat: method=satisfy
%%%mzn-stat: paths=0
%%%mzn-stat-end
```

Figure 3.1: Compilation statistics with Base model and $n = 500$

```
%%%mzn-stat: flatIntConstraints=1
%%%mzn-stat: flatIntVars=500
%%%mzn-stat: flatTime=0.106413
%%%mzn-stat: method=satisfy
%%%mzn-stat: paths=0
%%%mzn-stat-end
```

Figure 3.2: Compilation statistics with Global model and $n = 500$

The fundamental difference between using meta constraints and manual decomposition is that in the first case, the solver automatically creates these variables, while in the second case, the model has to do it manually. Thus, the slow compilation is mainly due to creating these Boolean variables.
We also noticed a big difference in propagation, as can be seen in Figures 3.3 and 3.2

```
%%%mzn-stat: failures=618
%%%mzn-stat: initTime=7.229
%%%mzn-stat: nodes=1246
%%%mzn-stat: peakDepth=251
%%%mzn-stat: propagations=359141208
%%%mzn-stat: propagators=500500
%%%mzn-stat: restarts=0
%%%mzn-stat: solutions=1
%%%mzn-stat: solveTime=12.502
%%%mzn-stat: variables=500500
%%%mzn-stat-end
%%%mzn-stat: nSolutions=1
%%%mzn-stat-end
Finished in 38s 590msec.
```

Figure 3.3: Solving statistics with Base model and $n = 500$

```
%%%mzn-stat: failures=989
%%%mzn-stat: initTime=0.003
%%%mzn-stat: nodes=1984
%%%mzn-stat: peakDepth=251
%%%mzn-stat: propagations=8924
%%%mzn-stat: propagators=2
%%%mzn-stat: restarts=0
%%%mzn-stat: solutions=1
%%%mzn-stat: solveTime=0.247
%%%mzn-stat: variables=500
%%%mzn-stat-end
%%%mzn-stat: nSolutions=1
%%%mzn-stat-end
Finished in 449msec.
```

Figure 3.4: Solving statistics with Global model and $n = 500$

This is because, in our view, the Base model having a very large number of constraints requires more propagation, instead in the Global model, the global constraint GCC has a single and global view on all the decision variables involved so consequently does not require all that much propagation. In fact we can see also the *solveTime* is very much more higher in Base model, precisely because of the reason explained above.

3.2 | Going from Base + Implied → Global + Implied

In this case, we can see that the table shows that the failures of the two models (with the addition of the implicit constraints) are practically equal. We also note that as n varies, the **Base model + Implied** has two more failures than the **Global model + Implied**, differently from the models seen previously in which the Base model's failures were significantly fewer.

If we talk about the total execution time, as before, we notice that the Base model + Implied takes much longer than the Global model + Implied. This happens for the same reason as before in the case of the implementation without Implied.

While in the **base → global** comparison we had a substantial difference between both compile time and resolution time, in this case, comparing **base+implied → global+implied** the difference is exclusively in compile time. In fact, the resolution time in the second case still remains very low and there is no substantial difference between **base+implied** and **global+implied** as we can see in the figures 3.5 and 3.6. This, in our opinion, is because thanks to the implied constraints both models have more useful information about the problem and as a consequence, the propagation performed is much more efficient.

```
%%%mzn-stat: failures=495
%%%mzn-stat: initTime=6
%%%mzn-stat: nodes=994
%%%mzn-stat: peakDepth=252
%%%mzn-stat: propagations=4229254
%%%mzn-stat: propagators=8876
%%%mzn-stat: restarts=0
%%%mzn-stat: solutions=1
%%%mzn-stat: solveTime=0.356
%%%mzn-stat: variables=500500
%%%mzn-stat-end
%%%mzn-stat: nSolutions=1
%%%mzn-stat-end
Finished in 24s 556msec.
```

Figure 3.5: Solving statistics with Base model + Implied and $n = 500$

```
%%%mzn-stat: failures=493
%%%mzn-stat: initTime=0.004
%%%mzn-stat: nodes=991
%%%mzn-stat: peakDepth=251
%%%mzn-stat: propagations=4962
%%%mzn-stat: propagators=3
%%%mzn-stat: restarts=0
%%%mzn-stat: solutions=1
%%%mzn-stat: solveTime=0.085
%%%mzn-stat: variables=500
%%%mzn-stat-end
%%%mzn-stat: nSolutions=1
%%%mzn-stat-end
Finished in 282msec.
```

Figure 3.6: Solving statistics with global model + Implied and $n = 500$

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

The first of the implied constraints does not improve the result, and the second implied constraint is sufficient to express the same constraint. The first implied constraint improves the base model because the decomposition of the *Global Cardinality Constraint* does not have the full view of the sequence, whereas the global constraint can consider all values simultaneously. The *Global Cardinality Constraint* counts how many times the value v_j, assigned to the variable, appears in the sequence, and this count can affect the count of other values. Constraint decomposition, on the other hand, creates a separate constraint for each value, thus losing the global view of the problem. The first implied constraint does not improve propagation because it provides information that the *global-Cardinality* constraint already possesses due to the third parameter counting occurrences. Each O_j counts how many times v_j appears in the sequence, thus providing the cardinality of X_i (where X_i is assigned to v_j). In fact, the GC constraint, can collect the counts and limit the number of counts to the number of occurrences of v_j and thus can already know all the information of the first implied constraint.