# N-QUEENS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***value ordering heuristics of Gecode*** | ***n*** | | | |
| *30* | *35* | *45* | *50* |
| ***number of failures*** | | | |
| input order – min value | 1,588,827 | 2,828,740 | - | - |
| input order – random value | 9 | 10 | 6 | 42 |
| min domain size – min value | 15 | 21 | 6 | 123 |
| min domain size – random value | **1** | **0** | **1** | **10** |
| domWdeg – min value | 15 | 21 | 6 | 123 |
| domWdeg – random value | **1** | **0** | **1** | **10** |

Tests were conducted with different values of N using various variable and value ordering heuristics in the context of the n-regine problem solved with the Gecode solver. The table shows the numbers of failures, with the best results highlighted in bold.

## Analysis and comparison of heuristics

In the table, it can be seen that Input Order - Min Value has a high number of failures, which indicates a less efficient search.

The choice of selecting variables in the given order and testing the lowest value first has the worst result in terms of failures. The reason is that selecting variables in the order of the input forces one to have no choice as to which variables are best selected first.

Therefore, if the first decision is wrong and the solver ends up in a large subtree, it can take a long time to get out of it and therefore has a long execution time.

In fact, for some instances, the solver takes much longer to find a solution, probably due to a particular ordering of the variables. Furthermore, we note that for instances where n is 45 and 50, the solver is unable to find a solution within the required time (5 minutes).

Looking at the table as a whole and comparing the heuristics with min value and those with random value, we notice that by using randomisation, the results improve. By examining different parts of the tree, the probability of finding a solution increases, thus reducing the risk of getting stuck in a subtree.

Furthermore, we note that the two heuristics min domain size and dowWdeg with both min values and random values exhibit the same failures.

This occurs because each variable is included in all the constraints, thus leading to initially equivalent weights for all variables. When the solver encounters a failure, the weight associated with the constraint is increased. However, since all variables share identical constraints, the weight of each variable simultaneously increases by 1. Consequently, the weights remain constant throughout the execution, and the only determinant in the selection of variables becomes the size of the domain.

In short, we can say that the criterion for selecting values is based on the ratio of the domain size to the weighted rank of each variable in the variable selection. This criterion favours the selection of variables with smaller domains and higher weighted degrees.

# POSTER PLACEMENT

## ORIGINAL ORDER

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***value ordering heuristics of Gecode*** | Data files | | | | | |
| 19x19 | | | 20x20 | | |
| ***number of failures*** | ***time*** | ***number of failures*** | | ***time*** |
| input order – min value | 1,362,457 | 2s 962msec | - | | - |
| input order – random value | - | - | - | | - |
| min domain size – min value | 239,954 | 4s 718msec | **1,873** | | **446msec** |
| min domain size – random value | 2,929,153 | 55s 830msec | 5,797,312 | | 1m 42s |
| domWdeg – min value | **236,024** | **6s 25msec** | **1,873** | | **483msec** |
| domWdeg – random value | 2,929,030 | 54s 109msec | 5,797,456 | | 1m 38s |

## ORDER BY DECREASING PERIMETER

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***value ordering heuristics of Gecode*** | Data files | | | |
| 19x19 | | 20x20 | |
| ***number of failures*** | ***time*** | ***number of failures*** | ***time*** |
| input order – min value | **30** | **176msec** | **323** | **183msec** |
| input order – random value | - | - | - | - |

We calculated the results we entered in the table with the original order of the values and then reordered the rectangles in the data file in descending order according to their perimeter.

Analysis and comparison of heuristics

The input ordering heuristic shows long execution times and frequent failures. This technique proves to be inflexible and encounters difficulties in identifying advantageous variables in the early stages of the search.

Like the previous problem, the first table shows practically equivalent results between the 'minimum value' and 'domWdeg' heuristics, although the latter sometimes produces fewer failures. This disparity is attributed to the presence of different constraints for rectangles. In particular, placing larger rectangles is more complex due to more stringent constraints than those associated with smaller rectangles. This variation in the constraints and size of rectangles contributes to a significant difference, as rectangles that are more difficult to place generate more frequent failures due to the greater rigidity of their constraints and the resulting greater weight. Therefore, they are preferred in the initial choices. In some scenarios, this dynamic makes the 'domWdeg' heuristic slightly more effective than the 'smallest domain' criterion, although both are outperformed by the static heuristic when the data are reordered.

In the context of this problem, the application of randomness never leads to improvements; on the contrary, it increases the number of failures. The position of the posters is strongly influenced by the stringent constraints of the problem. Consequently, using a deterministic approach, which follows the constraints, produces better results than randomisation.

In the second table, after the decreasing re-sorting of the posters, a significant decrease in failures is observed in the first variable-sorting heuristic. This occurs because the first posters have more stringent constraints, and placing them initially is advantageous. In the event of failure, this occurs immediately, preventing subsequent constraints from causing further failures after the placement of more posters. Similarly, smaller posters with less stringent constraints are more easily placed later in the execution.

Despite the fact that the addition of randomness continues to produce no improvement, even going so far as to exceed the time limit in the search for this heuristic.

We can conclude by stating that the decreasing reordering of rectangles is the main and most relevant feature of 'Poster Placement', as it allows the search to be scheduled for better performance.

# QUASIGROUP COMPLETION PROBLEM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data files |  | default search | domWdeg – random value | domWdeg – random value + restarting |
| *30-03* | ***number of failures*** | - | 1,061,184 | **642,427** |
| ***time*** | - | 2m 38s | **1m 42s** |
| *30-05* | ***number of failures*** | 657,955 | **5,885** | 303,205 |
| ***time*** | 1 m 33s | **1s 217 msec** | 45s 627msec |
| *30-08* | ***number of failures*** | **627** | 6,403 | 11,990 |
| ***time*** | **339msec** | 1s 180msec | 2s 454msec |
| *30-12* | ***number of failures*** | 259,082 | 53,200 | **21,986** |
| ***time*** | 32s 517msec | 8s 0msec | **3s 867msec** |
| *30-19* | ***number of failures*** | 381,330 | - | **48,244** |
| ***time*** | 54s 790msec | - | **8s 880msec** |

The table above shows the number of failures and the total time required by different search types for the quasigroup completion problem.

## Analysis and comparison of heuristics

The only case in which the default search method is more efficient is case qc30\_08, where the solver completely governs the default search. Probably some instances of the default search are affected by some bad decisions made at the beginning of the tree. The default search is completely dependent on the solver, so it could happen that it makes a wrong decision at the beginning of the search, gets stuck there and requires a lot of search time to undo it. Instead, in this case the first assignments provided by the default search are good, so opting for the most advantageous decision at the start of the search decreases the chances of failure and speeds up the path to resolution.

The domWdeg heuristic with random value presents the best result for case qc30\_05.

The domWdeg - random value + restarting heuristic presents better results for several data files.

In most cases, domWdeg + restart method improves performance because the solver has some previous information that it can exploit to find a solution faster and with fewer failures.

The method generates additional information (i.e. weights) during execution; therefore, when the search is restarted from the initial node, the solver (based on the previously generated information) selects the next variables, improving the search and trying to minimise the initial errors in the tree. In fact, restarting the search allows the solver to have the opportunity to make different and better decisions.

In this context, we observe that the restart does not prove beneficial for instances 05 and 08, presumably due to the random ordering of the variables, which is already optimal. In detail, this circumstance could explain why the solver initially does not spend the necessary time to explore the first branch during the restart phase, thus losing the benefit of the already optimal ordering of the variables.

Probably the reason is that the default search and the domWdeg heuristic already have a good ordering of the variables for these instances, which can produce a good solution. Therefore, if restarting the instance would change the good ordering of the assigned variables, the search could change completely or worsen.

When are random decisions (not) useful? Why?

Depending on the specific context, the introduction of randomness can be advantageous or disadvantageous.

In the nQueens problem, the addition of randomness brings benefits: randomly assigning a position to a queen on the chessboard is more advantageous than systematically placing it in the upper left corner. This is because the random placement of the first queens increases the probability that they control a larger number of squares than the minimum (top left) placement. Although this approach makes the subsequent placement of queens more complex, it helps to reduce the search space in the event of failure.

Random decisions are not useful in the poster placement problem, as random variables do not improve the results of the model. The random placement of a poster within the grid increases the probability of failure in the first steps, especially when it is a large poster.

Finally, in the quasi-group problem, the introduction of randomisation proves useful, especially when combined with the restart of the search.

Are dynamic heuristics always better than static ones? Why?

Dynamic heuristics are not necessarily an improvement over static heuristics. On the contrary, we have observed that in some circumstances, rearranging instance variables with problem-specific heuristics and subsequently solving with static heuristics yields more satisfactory results. Nevertheless, when no problem-specific heuristics are available, dynamic heuristics prove to be more useful as they possess the advantage of adapting to the current state of the search tree, thus being able to react more flexibly to specific challenges encountered during the resolution process. However, it is important to note that the effectiveness of this adaptability depends on the problem and the nature of the solution. In some cases, static heuristics may be sufficient and more efficient, especially if the characteristics of the problem remain constant.

Is scheduling the search and/or restart always a good idea? Why?

Programming search and restarting are strategies used to improve the effectiveness of research. Restarting can be useful to avoid getting trapped in an unpromising part of the search space. When both randomisation and restarts are present, they can eliminate the huge variance in solver performance. However, the effectiveness of these strategies is highly dependent on the specific configuration of the problem and algorithm. The appropriate use of restart strategies requires detailed evaluation and experimentation in the context of the problem addressed.

The section on REBOOT provides guidance on the possibility of restarting the search to correct initial errors that may cause a large amount of unnecessary searches. Restarting can be beneficial to avoid getting trapped in non-productive regions of the search tree. However, it is also noted that the success of the restart depends on the underlying search strategy and the presence of randomness to introduce significant variations.

The problem arises when wrong decisions are made at the beginning of the tree and this causes an exponential amount of searches to be cancelled. There is a solution to this problem: restart the search from the beginning to make different decisions. Several restart strategies exist in Minizinc: constant restart, geometric restart and Luby restart.

In the quasigroup completion problem, we note that the results of the search with restart can sometimes be worse than those of the default search. In particular, restarting does not always seem to be a good idea for some instances; this can be attributed to the fact that the default search and the domWdeg heuristics already present an efficient ordering of variables in some instances, which could lead to better solutions without the need to restart the search process.

“You have done much better this time. Did you write all the text yourselves (in which case your English seems quite advanced) or did you use a tool to support your writing? “

Thank you for your feedback. Yes, we write the texts ourselves but we confess that, despite our best efforts, we often find it difficult to write in English. Using a support tool helps us to translate concepts understood in our mother tongue into English. We would like to emphasise that, despite these language difficulties, we are working hard to achieve results in this subject, in fact we have asked you to make a call just to prove it to us. We know that English is important and we are trying to improve in this as well.

Thank you also for the correction of terminology. We have corrected them in our report.

“PP

The following sentences still do not explain why random decisions do not work well in PP. Note that there is nothing like random variable. The assigned values are chosen randomly. Can you try again to explain this?

"In the context of this problem, the application of randomness never leads to improvements; on the contrary, it increases the number of failures. The position of the posters is strongly influenced by the stringent constraints of the problem. Consequently, using a deterministic approach, which follows the constraints, produces better results than randomisation."

"Random decisions are not useful in the poster placement problem, as random variables do not improve the results of the model. The random placement of a poster within the grid increases the probability of failure in the first steps, especially when it is a large poster."”

Let us try again to explain why random decisions do not work well in Poster Placement (PP).

Variables are not chosen randomly in the context of the problem, but the values assigned to these variables are chosen randomly. In this context, randomly assigning values to variables does not improve the results of the model (i.e. does not lead to improvements).

The random placement of a poster in the grid increases the probability of failure in the first steps, especially when it is a large poster. In particular, the placement of a larger rectangle is more complex, as the associated constraints are more stringent than for smaller rectangles.

This variation in the constraints and size of the rectangles contributes to a significant difference. Rectangles that are more difficult to place tend to generate more frequent failures due to the greater rigidity of their constraints.

For these reasons, in the context of Poster Placement, it is crucial to follow a deterministic approach that respects constraints, as this tends to produce better results. By prioritising the search for posters that violate restrictions instead of starting with a random placement, we can minimise the risk of violating restrictions later in the resolution process, thus contributing to a more efficient and accurate solution.

“QCP

Which search method is the most robust overall? Qual è il metodo di ricerca più robusto in assoluto?”

The most robust search overall seems to be the domWdeg - random value + restarting method because it has better results for different data files.

In most cases, this method improves performance because the solver has some prior information that it can exploit to find a solution faster and with fewer failures.

The method generates additional information (e.g., weights) during execution; therefore, when the search is restarted from the initial node, the solver (based on the previously generated information) selects the next variables, improving the search and trying to minimize initial tree failures. In fact, restarting the search allows the solver to have the opportunity to make different and better decisions.

It is noted, however, that restarting does not prove beneficial for instances 05 and 08, presumably because of the random ordering of the variables, which is already optimal.

Some corrections in terminology:

•⁠ ⁠Each variable initially includes all constraints -> each variable is included in all the constraints

•⁠ ⁠Error -> failure

•⁠ ⁠Search/Research scheduling -> programming search (research is a scientific investigation)