Decision Making with Constraint Programming

Exercise 5

Dessì Leonardo [0001141270]

Spini Riccardo [0001084256]

Corso di laurea magistrale in Informatica

Alma Mater Studiorum
Università di Bologna

December, 2023

1 | nQueens problem

The table above shows the failures and the objective value added to the alldifferent model for the nQueens problem to find the best solution that approaches the main diagonal of the chessboard.

	failures	**solutions**	**obj**
default	16 149 996	5	810
domWdeg - rand	17 691 529	20	732
domWdeg - rand + restart	14 876 759	16	672
domWdeg - rand + restart + LNS	**4 429 668**	15	**650**

Table 1.1: Result of nQueens

To calculate the sum of the distance between queens and the main diagonal of the checkerboard, we used the following formula.

$$\sum_{\mathbf{i=1}}^{\mathbf{N}} |\mathbf{q}[\mathbf{i}] - \mathbf{i}|$$

where $|q[i] - i|$ represents the absolute difference of the position of a queen in column i and the number in that column, iterated for all columns.
In the main diagonal of a chessboard, the position of each queen is such that its column ($q[i]$) is equal to its index (i). Therefore, when $q[i]$ is equal to i, the difference $|q[i] - i|$ is zero. When a queen is not on the main diagonal, this difference will be different from zero and represent the horizontal distance between the queen and the main diagonal.
By minimizing this sum, we are trying to place the queens as close as possible to the main diagonal.
In our case, the objective function measures how close the queens are to the main diagonal. While the objective value refers to the numerical result associated with an objective function. In our script, we called the objective value *dist_diagonal*, representing how well you achieved the goal of placing the queens as close as possible to the main diagonal. So the lower the objective value, the better our solution method will be.
Remember that the heuristic *domWdeg* estimates a ratio between the domain size of a variable and the weighted degree, which is the number of times the constraints involving the variable fail. Then select the variable X_i with lowest $dom(X_i)/w(X_i)$. The weighted degree for each variable starts from 1 and changes dynamically during the search.
The *default* search method proves to be more efficient in terms of failures than *domWdeg - rand*. The *domWdeg* heuristic allows the solver to fail as soon as possible without getting

stuck in some sub-tree, but this leads to a much larger search and thus more failures than the *default* search.
In our opinion, it is important to say that this happens because we have given a time limit (5 minutes); if we had not given a time limit, the search *domWdeg - rand* would probably have fewer failures than the search *default*. This is because during the time limit, *domWdeg - rand* finds many more solutions than *default*, and consequently explores the search tree much more than *default*. Furthermore, as we mentioned in exercise 3, the random initial placement of the queens increases the probability that they check more boxes than the placement with *minvalue* (top left).
To test our hypothesis, we can run our script with fewer queens so that the program ends without giving a time limit. These are the results of the failures:

	default	**domWdeg - rand**
n=12	74 828	74 363
n=15	6 760 724	5 590 710

Table 1.2: Number of failures for 12 and 15 queens in the chessboard.

As for the *domWdeg - rand + restart* method, restarting the solver produces an improvement in this model and the reason is due to a better exploitation of some previous information, such as the consideration of the ratio between the domain size of a variable and the weighted degree, which allows us to restart the search by first choosing those with the lowest degree (obtained from the previous exploration of the tree).
Instead, *LNS* is a local search-based optimization technique that explores a larger set of solutions than a standard local search. In general, the concept of *Large Neighborhood Search* involves the exploration of larger "neighborhoods" of close solutions, permitting local minima to be passed and the solution space to be explored more effectively. The LNS views the exploration of a neighbourhood as the solution of a sub-problem and use tree search to exhaustively but quickly explore it.
The percentage of fixed variables is 85%, this allows a subtree equivalent to 15% to be explored for each restart. The few remaining variables are much easier to explore, as is the propagation of constraints.
Finally, a graphical representation of a solution to the nQueens problem was created with Minizinc, modifying a few things from the default `html` template to customize the grid colors. In light blue, the queens are represented, while in purple the diagonal is highlighted

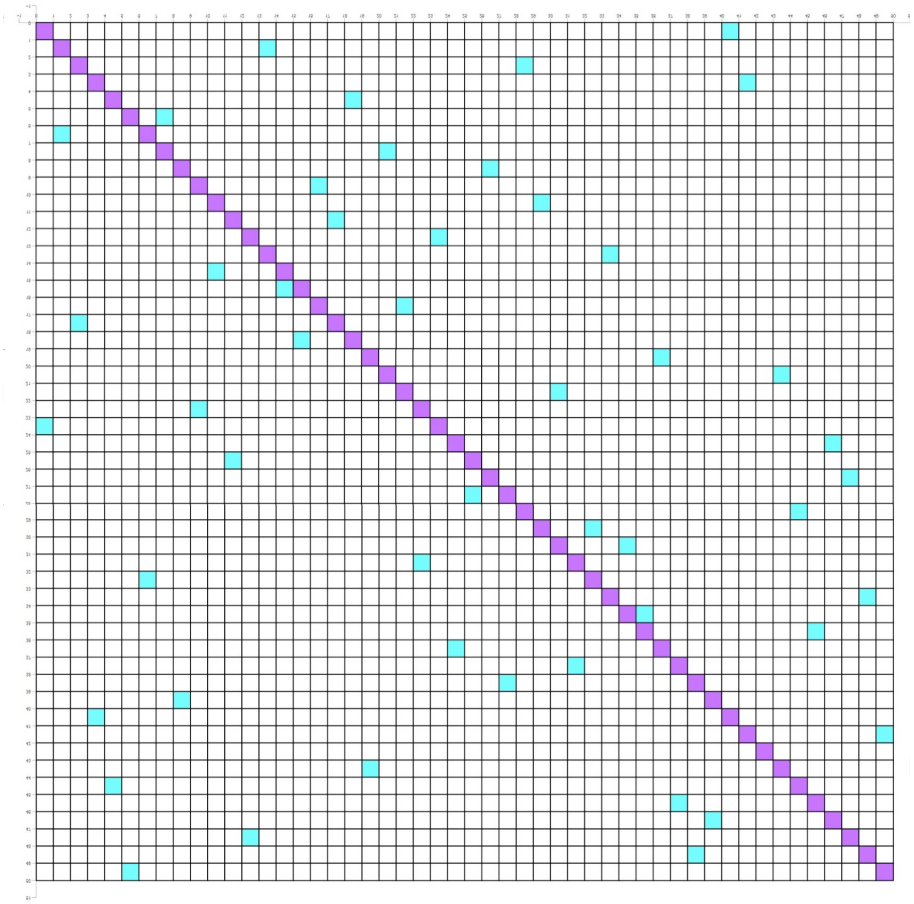

Figure 1.1: Graphical representation of the solution to the nQueens problem with $n = 50$.