Decision Making - ex 3

Filippo Brajucha, Youssef Hanna

November 2023

1 nQueens

1.1 Table

		30	**35**	**45**	**50**
input order	min value	1.588.827	-	-	-
	random value	9	10	6	42
min domain size	min value	15	21	6	123
	random value	**1**	**0**	**1**	**10**
domWdeg	min value	15	21	6	123
	random value	**1**	**0**	**1**	**10**

Table 1: nQueens problem with n = 30, 35, 45, 50

1.2 Comment

(ii)To begin within the exercise two ways of constraining a variable are considered are **Indomain min** (min value) and **Indomain random** (random value).The **Indomain min** give the variable its smallest domain value. To better explain this concept it was thought appropriate to quote this example from the MiniZinc documentation.

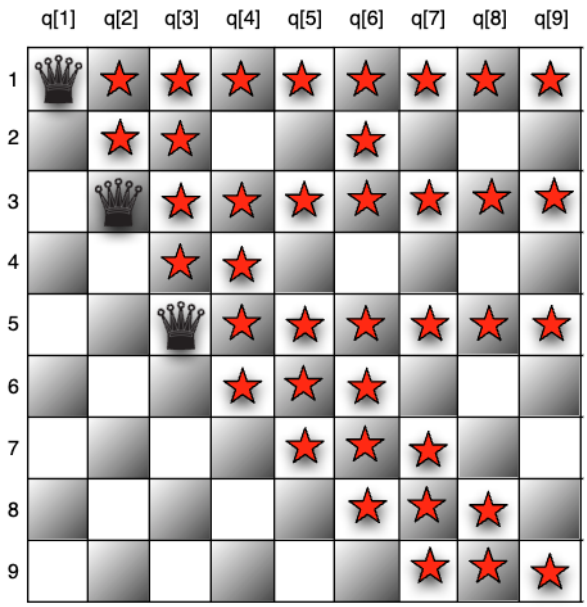

Figure 1: N-queens example

First of all suppose we use the first fail, then take the variable with the smallest domain, in this case q(6), and if we were to use the **Indomain min** we would have to take the smallest value that can be assigned to it, in this case 4, so q(6) = 4. As for **Indomain random**, the value of the variable would be chosen randomly, so if we always consider the figure above it could be q(6)=4 or q(6)=9 with the same probability. **Indomain Min** proves to be a strategy with more failures regarding **Min Domain Size** and **DomWdeg** for the same reason. Because the selection of variables is done by choosing those with the smallest domain. According to this variable selection strategy, it is more likely to choose variables on two neighboring columns because the neighboring columns will have the smallest domain for diagonals. So this strategy with the **Indomain Min** will also take those neighboring ones in rows as well as in columns. In this way each time a value is assigned to a variable the domain of values will be smaller and smaller forcing the choice of some values until a constraint is violated. If we take the figure from before as an example and implement this strategy, the next decision would be q[6] = 4. If we make this decision, there is only one value left for q[8], q[8] = 7, so this is forced, but then there is only one possible value left for q[7] and q[9], which is 2. So a constraint must be violated. We have detected unsatisfiability and the solver must go back by canceling the last decision q[6] = 4 and adding its negation q[6] != 4 . This removes some values from the domain, and then the search strategy to decide what to do is repeated. The **Indomain Min** also causes many failures in the Input Order strategy that

chooses variables in order from the array. If the order of the variables is strictly close it will cause the same problem as described regarding **Min Domain Size** and **DomWdeg**.It therefore depends very much on the order of the array.In this scenario, making backtracking very costly. So in this case, it is less likely that by randomly assigning values to variables, with **Indomain Random**, they will violate the constraints and be far in the checkerboard. A random approach would potentially allow the queens to already be in a correct position.

To explain why the min domain size and domWdeg results are the same when they use the same value ordering, it's because domWdeg utilizes both the minimum domain size and the weighted degree. During propagation, every time a constraint fails and becomes harder to satisfy, its weight increases by 1. Then, we choose the variable with the maximum weighted degree and the minimum domain size. Regarding the domain size, we select the variable with the minimum domain size. (iii)The explanation lies in the fact that all domWdeg variables are equally associated with all constraints, meaning they must satisfy all of them. Therefore, the solution is the same as the minimum domain size. So, in the mathematical formula of domWdeg, the denominator will be a constant, allowing the formula to be like that of the min domain size.

(i)With **Dynamic Heuristics** (Min Domain Size and DomWDeg) you get fewer failures, this is because the choice of variable is defined by the previous ones and this allows a more optimizing choice with fewer backs. Whereas with Input Order using a **Static Heuristic**, that is, the order of choice of variables is already predefined by the array, it is more difficult to satisfy the constraints since the order of variables is independent of each other.

2 Poster Placement

2.1 Table

		19x19		20x20	
		Fails	Time	Fails	Time
input order	**min value**	**1.362.457**	**9s 332ms**	-	-
	random value	-	-	-	-
min domain size	**min value**	239.954	2s 64ms	**1.873**	**291ms**
	random value	2.929.153	19s 173ms	5.797.312	37s 792ms
domWdeg	**min value**	236.024	2s 6ms	**1.873**	**290ms**
	random value	2.929.030	19s 752ms	5.797.456	37s 169ms

Table 2: Poster Placement using 19x19.dzn and 20x20.dzn (unsorted)

2.2 Comment

With **Min Domain Size** and **DomWDeg**, we obtain the best results as observed from the table. This is because in both cases, we choose variables with the smallest domain size. As for DomWDeg, even better results are obtained than Min Domain Size. These results are due to the fact that we choose the variable with the smallest value of domain size divided by weighted degree, which is the number of times it has been in a constraint that caused failure earlier in the search. In this way, pieces that are larger will have more difficulty meeting the constraints so they will be chosen first.

Random assignment is much less efficient because there is a risk of placing large pieces in the center of the board. This makes it difficult to place other pieces because the space is divided poorly. On the other hand, with the min value assignment, the placement starts from the left, improving space management.

2.3 Table with sorted rectangles

	19x19		20x20	
	Fails	Time	Fails	Time
min value	**62**	**139ms**	323	153ms
random value	59.743.367	-	59.747.616	-

Table 3: Poster Placement using 19x19.dzn and 20x20.dzn (sorted)

2.4 Comment

As can be seen in Table number 2, the static heuristic with a sorted array significantly improves the statistics. Variables are chosen based on the sorting of the array, allowing the placement of larger pieces first. Placing larger pieces first is advantageous for the nature of the problem because it is easier to fit in larger pieces that occupy more space and then slot in the smaller pieces. On the other hand, placing smaller pieces first and then having to fit in a larger piece has a much higher probability of not fitting because it is a single compact piece, and there might be smaller pieces scattered around.

3 Quasigroup

3.1 Table

		default	**domWdeg** - random	**domWdeg** - random + **Luby**
qc30-03	Fails	-	1.061.184	**642.427**
	Time	-	2m 11s	**1m 23s**
qc30-05	Fails	657.955	**5.885**	303.205
	Time	1m 27s	1s **366ms**	39s 733ms
qc30-08	Fails	**627**	6.403	11.990
	Time	**600ms**	1s 283ms	1s 794ms
qc30-12	Fails	259.082	53.200	**21.986**
	Time	32s 430ms	7s 852ms	**3s 89ms**
qc30-19	Fails	381.330	-	**48.244**
	Time	52s 450ms	-	**6s 876ms**

Table 4: Quasigroup problem resolution with using *qc30-03.dzn*, *qc30-05.dzn*, *qc30-08.dzn*, *qc30-12.dzn* and *qc30-19.dzn*

3.2 Comment

Observing the table, it can be stated that the best solution to adopt when using a depth-first search is the domWdeg heuristic with the Luby restart. Search by restart is much more robust in finding solutions, as it can avoid getting stuck in a non-productive area of the search. The easiest way to ensure that something is different on each restart is to use some randomization, both in the choice of variables and in the choice of values. Alternatively, some variable selection strategies make use of information gathered from previous searches and thus will yield different behavior, e.g. **DomWDeg**. In our case we use both **DomWDeg** combined with a random variable selection, so using **Luby Restart** is definitely a good choice.

However, when observing qc30-05, the best result is achieved with the **DomWDeg - random** combination. This is probably because restarts occur too early in the search process, preventing sufficient exploration of the tree to make the restart beneficial. Another specific case is in qc30-08, where the default option performs better than the others.This happens because the domWdeg heuristic follows the principle of first fail, attempting where failures occur most frequently. The solver will fail a number of branches to reduce the search space of the tree. Moreover, the likely reason for the shorter search times in qc30-08 suggests that the instances to solve are simpler. Therefore, it could be inferred that the default search works better with simpler data compared to domWdeg, which is designed for more complex instances.