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

We can observe that the number of failures with the "input_order - min_value" is extremely high compared to other ones. It is in the order of the 10^6, other datas are in the order of 10^2; this is the only case where the system cannot perform the research for `n = 35, 45, 50`.
Generally we can observe that the random model is always better for each instance in terms of failures for this problem. The number of failures for `min domain size` and `domWdeg` with random value are the same and always better than the input order resolution.
We can imagine that the number of failures in the resolution with `input order - min value` is very high due the logic of the resolution of the problem, if we put all the queen in the minimum of the ordered domain we can be sure that there will be a lot of fails caused by the constraints of the problem (there cannot be quuen in the same row, line and diagonal). So, a lot of failures and a lot of excecution time (in fact for `n = 35, 45, 50` the solution isn't reached in time because of propagation and backtracking of a lot of constraints).

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 Table with sorted rectangles

	19x19		20x20	
	Fails	Time	Fails	Time
min value	**30**	**288ms**	323	466ms
random value	-	-	-	-

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

2.3 Comment

Examining mistakes related to assigning variables at random reveals a persistent limitation for our particular issue. The primary problem is that assigning random variables in this situation makes it more probable that constraints will not be satisfied. Due to the complexity of these limitations, variable assignment requires a more methodical and intentional approach. Due to previous random placements, random assignment may produce situations where pieces occupy sections of the poster, creating an absence of space for later pieces and resulting in constraint failures. Consequently, in order to guarantee variable assignment that consistently complies with problem constraints, a more methodical approach is required.

3 Quasigroup

3.1 Table

		default	**domWdeg** - random	**domWdeg** - random + **Luby**
qc30-03	Fails	-	**1.061.184**	**1.061.184**
	Time	-	**2m 11s**	**2m 11s**
qc30-05	Fails	657.955	5.885	**5.885**
	Time	1m 27s	1s 366ms	**1s 303ms**
qc30-08	Fails	**627**	6.403	6.403
	Time	**600ms**	1s 283ms	1s 574ms
qc30-12	Fails	259.082	53.200	**53.200**
	Time	32s 430ms	7s 852ms	**1s 569ms**
qc30-19	Fails	**381.330**	-	-
	Time	**52s 450ms**	-	-

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

The solver time-out is setted to 300 seconds (5 minutes), for each resolution heuristic the first dataset didn't arrive to a solution.
With all the resolution heuristics we can see that there are no big differences in terms of time, some seconds or some milli-seconds so all the methods use quite the same algorithms. The number of failures is always the same.
We can't determinate which of the method is the best because of sometimes the best method is the default one, other times is the worst. Observing the time datas we can say that using the restart strategy isn't a good idea but neither a bad one, the time and the failures are quitely the same, so maybe that is the same method that is used by the solver, without the annotation.

4 Questions

1. When are random decisions (not) useful? Why?
2. Are dynamic heuristics always better than static heuristics? Why?
3. Is programming search and/or restarting always a good idea? Why?

5 Answers

1. Measured data varies greatly depending on the nature of the problem; the only scenario where resolving with random heuristic is the best solution is with the nQueens problem.
   In the poster placement problem, it's always the worst solution, differing by orders of magnitude both in the number of errors (6 million errors compared to around 2 thousand) and in resolution times (several seconds compared to about 300ms). In the situation with ordered posters, it times out.
   Using the random heuristic with the quasigroup problem, that result to be the best solution 2 times out of 5. Due to this we can think that the input order is crucial in the poster placement problem.
   Random solutions, therefore, are not suitable in situations where maintaining an order in data usage is necessary. In fact, in both the quasigroup and nQueens problems, the choice of starting point isn't important since finding a solution requires starting from any point.
   The scenario where it becomes most apparent that the random heuristic is the worst is solving the nQueens problem when input order is specified. Using the input order heuristic means that each element, in order, is assigned the minimum value from the domain. The issue arises when it's necessary to find a solution; the solver encounters a situation where all elements do not satisfy the constraints. Consequently, it has to perform numerous backtracking operations to ensure that all points on the grid adhere to the constraints.

2. While it may be commonly assumed that dynamic heuristics are always superior to static ones. When applied to the Poster problem, a static heuristic works well. The reason is due to the nature of the problem itself: starting with the larger pieces of the poster and working your way down to smaller ones is the most efficient way to efficiently satisfy constraints. In this situation, using a static heuristic saves computational effort by removing the need to recalculate piece sizes at each iteration. Using a dynamic heuristic, on the other hand, would require extra computations because the best course of action is already known. Therefore, static heuristics can perform better than their dynamic counterparts in scenarios where a fixed, predetermined strategy works well.

3. The use of restart methods is a good idea when the problem is depth first search since by starting over from the beginning you have a better chance of finding the solution faster instead of continuing with the classic search (and with the constraints porpagation).
   The data we collected using restart in the quasigroup problem suggests that restart is not useful in solving this problem, the number of errors doesn't change, as the execution time which sometimes increases and sometimes decreases but always with a very little difference. So we can assume that there are no differences between domWdeg resolution and domWdeg resolution with restart, so the method use the restarting method by its own. This is probably a consequence of the fact that "indomain-random" is applied to the data anyway and the only method to resolve the problem is to restart when there's an error.