Team:
- Andrea Corradetti - andrea.corradetti2@studio.unibo.it
- Francesco Corigliano - francesco.coriglian2@studio.unibo.it

Summary

– Make observations on your results and write them down (avoid repeating the numerical results in text).
While doing so, pay attention to the following points:
– When are random decisions (not) useful? Why?
– Are dynamic heuristics always better than static heuristics? Why?
– Is programming search and/or restarting always a good idea? Why?

The first assignments are usually not well informed and result in exploring an unsolvable sub-tree.
This seems to be the case with the N-queens problem: when we place down the first queens, we don't know yet if the current moves will admit a solution.
Using a dynamic heuristic like minDomain (Fail First principle), we prove the infeasibility of the sub-tree more quickly.
Dynamic VOHs spend some time computing the order of variables, but unless we know an a priori optimal order (like in the Poster problem), they seem like the better choice.
Random value assignment can be a good heuristic when no further information is present to make an educated choice.

N-Queens

N	input order – min value	input order – random value	min domain size – min value	min domain size – random value	domWdeg – min value	domWdeg – random value
30	1'588'827	9	15 ~~35~~	**1**	15	**1**
35	2'828'740	10	21	**0**	21	**0**
45	-	6	6	**1**	6	**1**
50	-	42	123	**10**	123	**10**

- Input order - min value has the largest number of failures. For this particular instance, the strategy likely spends a lot of time in unsolvables subtrees due to wrong early decisions.
- Random values strategy is very effective: in the n-queens problem Input order - min value is an unfortunate strategy. We also don't have a lot of information at the very beginning to guide: the variables all have the same domains and the same number of constraints at the start, so the solver doesn't have a lot to go by. In this case choosing randomly can be a good alternative.

- minDomainSize can't help much at the very beginning but will soon become useful after the first assignment. The solver will try to satisfy the hardest variables first. These are the variables with the least available values and the ones that are more likely to become unsatisfiable due to constraint propagation from assigning something else.
- The minDomainSize / domWdeg strategies try to quickly prune the sub-trees that are most likely to fail so that they don't have to be explored in entirety.
- "min domain size - random value" and "domWdeg - random value" result in the same numbers. Small instances of the problem with the default random seed result in very few failures. Because the variables have the same number of constraints and the number of failures is small domWDeg behaves similarly to minDomainSize: the weights don't have a chance to change significantly.

- "Because the variables have the same number of constraints and the number of failures is small domWDeg behaves similarly to minDomainSize: the weights don't have a chance to change significantly." -> having the same number of constraints does not guarantee that the two heuristics result in the same order. Take into account the definition of domWdeg and think again.

Dom_w_deg = domain size of variable / sum of weighted constraints on variable.

In nqueens, the variables have the same number of starting constraints (horizontal, diagonal attack) and the same starting domain (1..n).
Possibly, because all the variables have the same global constraints on them, whenever one of them fails and its weight increases, the sum of the weights will still be the same for all variables. In the end, decisions will still be made according to domain size.
(Also, I corrected a typo in the table that threw me off last time)

Question for the professor: whenever we make a right branch and exclude a value from the domain of a variable, is this a new constraint being posed? And is this new constraint taken into account by dom_w_deg?
e.g. Say we backtrack and figure out that X != a. Is this a new constraint on x that could have its weight changed by dom_w_deg?

You need to however be careful with the definition of domWdeg. You wrote in the report "sum of weighted constraints on variable". We cannot sum the constraints smile Do you understand the dividing factor in domWdeg?

New addition:
Thank you for the correction. I see that the words were not precise.
It's the domain size of the variable / sum of the weights of the constraints on the variable.
All weights start at 1. The weight for a constraint is incremented by 1 each time it fails during propagation

Poster placement

POSTER PLACEMENT - default order

N	input order – min value	input order – random value	min domain size – min value	min domain size – random value	domWdeg – min value	domWdeg – random value
19x19	1'362'457 - 5,60s	-	239'954 - 0,939s	2'929'153 - 10,8s	**236'024 - 0,93s**	2'929'030 - 10,85s
20x20	-	-	**1'873 - 0,042s**	5'797'312 - 20,9s	**1'873 - 0,032s**	5'797'456 - 21,12s

POSTER PLACEMENT - decreasing order

N	input order – min value	input order – random value
19x19	**62 - 0,062s**	-
20x20	**323 - 0,047s**	-

- Here it makes sense to place down the largest posters first, which are also the hardest variables to satisfy. That's why decreasing order is useful.
- Choosing random values doesn´t make much sense: it will leave useless spaces between posters.
- Min domain size - domWdeg perform similarly with a slight edge for domWDeg. This could be a result of domWDeg adapting after a certain number of failures.
- In this problem, the default search with static heuristics (bottom up, left to right), tends to fail more because it doesn't optimize space as much as dynamic VOH (leaving blank spaces).
- “Min value” will try to squish the boxes towards the margins. Unfortunately, “input order” will pick small and large boxes according to the data file. This might result in small unused empty spaces. When choosing variables this way, the first few assignments will probably result in the solver getting stuck in an unsolvable sub-tree.
- Also, when backtracking the constraint posed will be of the like of X_i != a, which does not significantly reduce the searchspace.

- "Min domain size and domWdeg perform similarly with a slight edge for domWDeg. This could be a result of domWDeg adapting after a certain number of failures." -> you will understand why they are similar once you find out the correct explanation above

In this case, diffn is also a global constraint so a behavior analogous to nqueens is expected; that is, whenever diffn's propagation fails, its weight will be increased and this increase will be reflected in every variable's sum.
Here however we also have constraints on each variable. Those constraints might fail separately and change the weights for a single variable. We suppose this might explain the small advantage that dom_w_deg has over min_domain_size.

Quasigroup completion

QUASIGROUP COMPLETION				
	default search	domWdeg - random value	domWdeg – random value + restarting	domWdeg – min value + restarting
3	-	1'143'760 - 1m 22s	363'712 - 31,7s	**155'088 -**
5	562'850 - 43,5s	**4'823 - 0,47s**	224'804 - 19,27s	268'274 -
8	**1'491 - 0,184s**	3'291 - 0,32s	24'588 - 2,27s	79'815 -
12	241'500 - 16,19s	30'819 - 2,41s	**1'824 - 0,24s**	12'961 -
19	**281'622 - 22,22s**	-	319'632 - 28,36s	1'741'298 -

- We see that no strategy is all around more effective for every instance. This is probably evidence of the difficulty of the problem being influenced by the starting parameters and the search strategy.
  Indeed, default search won't be able to solve instance 3 but will quickly solve instance 8. Assuming that the bottom left of the tree corresponds to adhering to the heuristic, following that path will end up in an unsolvable tree for instance 3 but will quickly find a lucky path for instance 8.
- Restarting will at least find a result for every instance while the other 2 strategies can't find a solution within the time limit for one instance.
- Some areas of the grid might be harder to satisfy depending on the starting configuration. Using domWdeg + restarting the solver uses knowledge of past failures to figure out which variables are harder to satisfy and avoids getting stuck in an unsolvable sub-tree. In this way we are getting rid of an eventual heavy tail behavior.

Yes, here restarting based approaches are the most robust. Can you say why on instances 5 and 8, restarting could not be helping domWdeg+rand and default search, respectively?

Assuming default search is deterministic, restarting would result in the same exploration of the tree every time.
domWDeg+random finds a solution quite quickly for 5 and 8. It seems like restarting might stop the solver too early, preventing it from reaching those solutions.

You don't know if default search is determinist or not. In any case, what I wanted to ask was: on instance 8 for instance, even domWdeg-rand does not improve default search. We will often have such instances. Do you understand why this could happen?

new addition:
We think default search is an especially lucky heuristic for instance 8 as dom_w_deg + rand is for instance 5.
Restarting is reducing the variance as expected (we can see the other strategies struggle with instance 3), but does not guarantee an improvement. It helps to avoid getting stuck in exponential sub-trees for too long due to wrong early decisions (probably what's happening for default search in instance 3).
Dom_w_deg does not perform badly, it just has decent performance on all instances, while "default search" and "dom_w_deg +random value" get very lucky on a few occasions and struggle in others.

We talked in class about unfavorable combinations of heuristics and starting parameters that might cause unusually large solve times. This seems to be evidence of that.