DD2380 Artificial Intelligence: Homework 2, Part B, Answers

Mattias Mikkola, mmikkola@kth.se
Peter Boström, pbos@kth.se

September 15, 2010

N-Amazons

a) CSP constraints

- $row(a_i) \neq row(a_j)$
- $col(a_i) \neq col(a_j)$
- $diag\_ascending(a_i) \neq diag\_ascending(a_j)$
- $diag\_descending(a_i) \neq diag\_descending(a_j)$
- $pos(a_i) \notin knight\_moves(a_j)$

For all $i \neq j$.

b) write a program See *amazon.c*

c) method We're using a min-conflicts heuristic starting with a randomly permuted board. All pieces are placed with no horizontal or vertical conflicts. The pieces with the most conflicts are moved to the lowest-conflict position inside its column iteratively until all pieces are ok. Among the pieces with least conflicts one is picked randomly. This leads to more variation and doesn't get stuck as easily. Getting stuck in a local optimum is also an issue. That's why we count iterations, and unless the board is solved within n iterations, we shuffle the board and begin from the start.

This random element means that the board is not *guaranteed* to be solvable, but practically is. Larger boards seem to be easier as well. The random element also makes the solve time potentially unpredictable with bad luck. Most practical issues are accounted for. Run tests on this method seems to run in expected time $O(n^2)$, which is considered ok. It solves for $n = 10,000$ in about a second and for $n = 100,000$ about 2 minutes on this machine, so it's considered to scale pretty well. For larger values of n, this would of course become a problem.

The implementation however, even for n-queens, doesn't come near the book's example of n-queens in which this method would've solved a million-queens problem with 50 iterations.

A suggestion for improving the performance (not drastically) would be to cache more of the data, so that the number of conflicts doesn't have to be recalculated from the start each iteration.

d) performance This performance analysis should definitely be taken with a grain of salt. We chose not to do more calculations between 10 and 1000, since they're too fast either way. The script included (*tests.sh*) can be modified to run tests for other values as well, should it be desired.

n	iterations	time (s)
10	779	0.001
100	390	0.002
500	308	0.005
1000	507	0.013
2000	966	0.052
3000	1298	0.103
4000	1706	0.163
5000	2149	0.262
6000	2463	0.362
7000	2883	0.488
8000	3305	0.685
9000	3671	0.798
10000	4070	1.002