Assignment 4: Local Search

Brian Jacobel

Artificial Intelligence, Fall 2013

1 Introduction

In this lab, I implemented the local search algorithm using the MIN-CONFLICTS heuristic in a program designed to efficiently solve the "n-queens" problem. This constraint satisfaction problem describes the task of placing n queens on an n-by-n chessboard, and iteratively arranging them so that no queen is in a position of being able to capture another queen within a single move. In this problem, there are $6n-2$ constraints: n constraints specify that each column may have only one queen, n constraints specify the same for each row, $2n-1$ constraints specify that there may be only one queen in each upwards-sloping diagonal, and $2n-1$ constraints say the same for each downwards-sloping diagonal. The column constraints can be thought of as "freebies" – that is, queens can be obviously distributed at the start such that each is in a separate column, and this can be maintained throughout the problem easily. Thus, the variables that form the search space for this problem are simply the row locations for each queen.

2 Testing methodology

My algorithm is contained in the Python module `nqueens.py`, which can be run interactively via the command line if the user wishes. For testing, this code is imported into the script `test.py`, which runs the code a number of times and averages statistics. For each run of `test.py`, the n-queens problem is solved ten times each with with n starting at 5 and incrementing by 5 up to a maximum of 60. At each value of n, the average run time for the problem to be solved and the average number of moves required to find that solution is displayed. If one of the ten trials should fail, only successful trials are included in these values. However, in my tests, my program seldom failed to find a solution to the problem for a value of n less than 100.

I ran `test.py` six times: one for each of the variants of the algorithms explored in the assignment. In the upcoming section, I will display and discuss the differences in generated results between these algorithm variants.

As a point of comparison between my results and others', my testing was done on a fairly old Intel Xeon E5620 machine (a 2010 Mac Pro) with eight cores clocked to 2.40GHz. This CPU is somewhere in the neighborhood of 60% as fast as the newer i7-2600 CPUs in Searles 224's iMacs, according to benchmarks I found online.

3 Algorithm Variants

3.1 Basic implementation

For the basic local search algorithm with MIN-CONFLICTS as a heuristic, I obtained the results shown in *Fig. 1*.

3.2 Greedy implementation

3.3 Random implementation

In this algorithm variant, an unsatisfying queen is picked in the same way as in the basic algorithm. At this point, with some probability, instead of using the standard MIN-CONFLICTS heuristic to find a new location for this queen, this modified algorithm instead randomly picks a location in the queen's column to place it. In my trials, I found the probability 20% a good portion of the time to move randomly rather than heuristically; percentages higher than this resulted in the problem not being solved for a significant portion of the tests run. At 20% random movement, the constraints failed to be satisfied after 500 tries once with 35 queens and three times with 55 queens.

3.4 Comparison of above variants

4 Further variation

4.1 Smarter initial placement

4.2 With restarts

4.3 Random movements

5 Large values

My implementation of the MIN-CONFLICTS algorithm successfully solved the n-queens problem for an n of up to 350. Here are the n-values, moves and run times for various large values of n I tried. All tests below were done with the initial, unmodified implementation of the algorithm. Because of the times involved in obtaining the results, each was only run once.

I was slightly disappointed that my implementation of the algorithm could not get anywhere near solving even 1,000 queens in a reasonable amount of time. I do not know if this was a realistic expectation. I have declined to graph these values because of their uneven distribution, incompleteness, and lack of multiple trials to back them up – a graph would not reveal anything significant other than that this problem gets quite hard to solve with size.

n-value	moves to solution	time to solution(sec)
25	28	2.456
50	118	6.715
75	121	20.935
100	188	74.017
150	224	281.348
200	295	850.566
250	298	1637.862
350	407	8119.451 (2.255 hours)
500	N/A (hit max with 26 conflicts unfixed)	N/A
1000	never finished More than 18 hours	
1000000	never finished More than 18 hours	

6 Conclusion