School of Computing Science
Simon Fraser University

CMPT 310 - Introduction to Artificial Intelligence

Term: **Fall 98-3**
Instructor: **Bill Havens**
Office: **10828 APSC**
email: **havens@cs.sfu.ca**

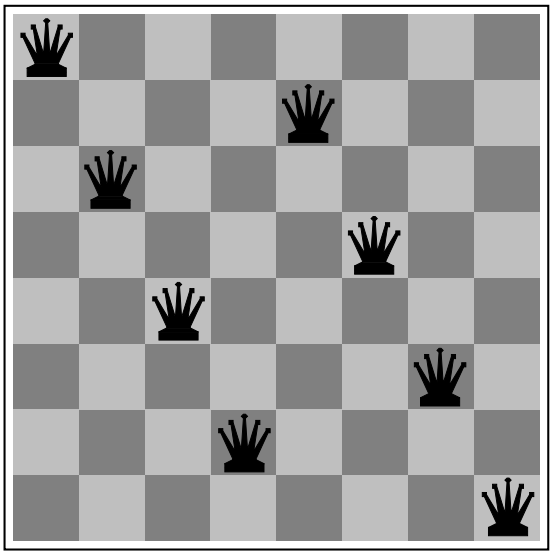
Assignment #3: Heuristic Search

Assignment

Heuristic search is a very active research area. What heuristic knowledge is appropriate for solving a particular application? What search methodology can make best use of available heuristic knowledge? In this assignment, you will explore a few types of heuristic search and compare it to constructive (backtrack) search.

We will explore these ideas using a toy problem, the n-queens problem. This problem used to be considered technically interesting. Various forms of constructive search and constraint satisfaction could solve n-queens for 10 or 20 or maybe more queens. But then heuristic search techniques demonstrated solutions for n = 1,000,000. Quite an improvement!

The n-queens problem consists of an n x n chessboard and n queens. You must place each queen so that it does not attack any other. Notice that no two queens can be on the same column of the chessboard. Hence it is easy to express this problem as a CSP where the columns correspond to variables, one for each queen and the rows correspond to domain values for the queens. The constraints are all binary between each pair of queens. Two queens are at “peace” if they are not on the same row or diagonal (and not the same column also of course). So the CSP graph is a clique with an arc between each pair of queens. Here is a solution to the 8-queens problem. There are solutions for every value of n except 2 and 3.



Homework

1. Implement the n-queens problem using heuristic repair methods (a.k.a iterative improvement). Use simple hill-climbing with the min-conflicts heuristic as your evaluation function. Define the predicate, `peace(Q1, Q2)`, which returns true if queen Q1 and Q2 are not on the same row or diagonal and false otherwise.

   Define a move to be the exchange of rows for two different queens on the chessboard. Generate a random position of queens on the chessboard to begin the search. Return the first consistent solution.

   What happens if the search gets stuck at a local minima (*i.e.* no move reduces the number of violated constraints for min-conflicts)? Put your hill-climbing scheme in a loop which randomly resets the starting point of the search after say 100 moves. Does this work well?

   Test your hill-climber on n = 10, 100, 1000, 10000 queens. Can it do that many?

2. Implement the n-queens problem using simulated annealing. Again use the min-conflicts heuristic as your evaluation function. Define a "cooling schedule" for your search which reduces the probability of a random move as the number of cycles in the search increases. Usually such schedules decay rapidly at first and then asymptotically approach zero. Try various cooling schedules until you find one which works well with n-queens.

   Now test your simulated annealer on n = 10, 100, 1000, 10000 queens. Maybe it can do more. Compare its performance (using your tailored cooling schedule) to hill-climbing with random restart in part-1 above.

Due Date: Monday, November 2 before 3pm in the class assignment box (note change of due date)