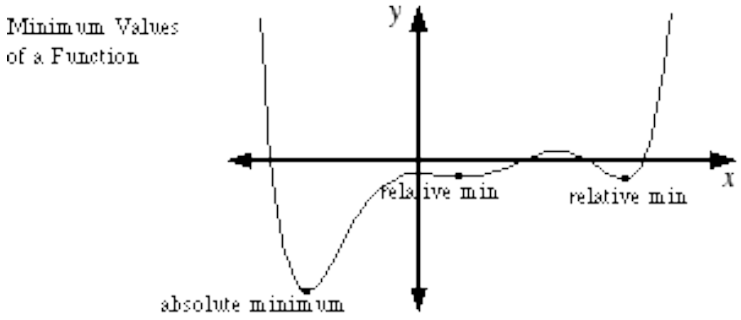
Tristan Adams
HW3

1. Heuristics for CSP
    a. The value which is most constrained will be impacted the most by following decisions, and a variable which is least constraining imposes the fewest constraints. Combining these means that the variable can be altered more easily than others, since it is not highly constraining, but at the same time it can be used to measure success, as it is the most constrained. If a solution can be found for a variable altering a variable which is most constrained, the solution will therefore work on other variables which are not as constrained.
2. Magic Squares and N Queens
    a. Code is submitted to the directory in ms.py and nq.py
        i. Ms.py is adapted from ms3.py and ms4.py but is modified to be able to accept any N and find its magic square value ($N*(N^2+1)/2$)
        ii. Nq.py is adapted from the examples of using the constraint package. It loops through the columns and adds a constraint based on a function which takes in the rows and columns from the nested loop containing it, and makes sure that the board cannot have a queen in the same row, column, or diagonal as another.
    b. The MinConflictsSolver uses Minimum Conflicts Theory, which adds to solutions based on how few conflicts the action would have. So, if the initial state of the board is not just right, the solver can fall into a loop, since it goes into a local minimum of conflicts in some problem, when there could be a lower absolute minimum. The below picture illustrates local minimums vs actual minimums. Even with that, though, MinConflictsSolver resulted in some of my better runtimes for the N Queens problem. One run I had was even able to find a solution for the 256 Queen Problem before the 30 second timeout, solving in a little over 28 seconds. However, the MinConflictsSolver failed on all types of Magic Squares problem. This is due to the nature of magic squares, which uses sum constraints. Each time a decision is made, that decision affects all of the others which prevents the solver from successfully finding a solution.
        i. Picture of Local vs Actual Minimum:



Picture is from http://www.mathwords.com/a/absolute_minimum.htm

    c. With each increment in N, the problem grows exponentially. A magic square has N^2 units. There are 9 boxes in a magic square of size 3, 16 in one of size 4,

25 in one of size 5, and so on. Since the area to search increases so quickly, it takes that much longer to search and solve these grid problems. Watching the runtimes for the magic squares solvers, the time the various solvers take between increments of N seems to jump at an extremely high rate, possibly exponential.

None of the solver types in test.py could find a solution within 30 seconds before timing out. However, when run without a time limit, and waiting for 10 minutes, ms.py was unable to provide output for getSolution(). This is due to how the board size is N^2 and how constraining it is each time a new decision is made. With the high constraining done by each decision, the amount of time a solver takes to find a solution becomes huge with small increases to N.

This goes back to Question 1; this problem is not good for solving the Magic Squares problem because each decision is highly constraining, when the better type of heuristic to use is one that is most constrained but least constraining. The Magic Squares problem is too highly constraining to properly fit the CSP methods.