Timothy Mason UMSL ID: thmmqc
millionsofbillions@gmail.com
Graduate student (graduated in December 2010)
Major: Physics
Command to create executable : make
I agree to attend the ceremony.

This algorithm employs an optimization approach. A heuristic is calculated by summing the numbers in each column, row, and diagonal, and subtracting from the magic square number M. The sum of the absolute values of each these sums form the final heuristic that determines how far the solution is from being a magic square. A heuristic of zero implies that a magic square has been found.

The solution evolves using a greedy algorithm that accepts only changes to the solution that lower the heuristic. The square is initialized filling one row at a time by simple counting except for the constrained squares. This is both easy to implement, and is intentionally very far from a magic square solution with the hopes that it therefore has access to more possible magic square solutions (that is, does not start near a local minimum trap). A move in the solution is accomplished by choosing two squares at random and swapping their contents. However, instead of determining convergence by swapping with no improved solution for a predetermined amount of moves, the entire space of possible swaps is surveyed ensuring that if a lowering move exists, it will be found. Convergence is determined only after a complete survey has been conducted with no improvement. The randomness of the process is introduced only by choosing a random starting position in the search of the swap space. Once convergence is detected, the algorithm stops if it has found a solution, or else it re-initializes and tries again.

No evidence was found that it was effective to continue the search from a local minimum non-solution and accept a number of random swaps that increase the heuristic before descending again. Total re-initialization seems to be the best strategy. At the top level, the greedy algorithm finds a solution X percent of the time, and it's just matter of trying it repeatedly until it finds a hit. Chart A shows a plot of the number of attempts required on average as a function of the degree and Chart B shows the increase in average time required. As the time required falls into a random distribution, luck is required at the higher degrees to score a win in the competition. Degree 15, for example, will find a solution within 60 seconds about 65 percent of the time for the unconstrained case. There is therefore a 27% chance that the algorithm will score at this level by finding a solution under 1 minute 4 out of 5 tries.

Constraints are dealt with simply by placing the constraint matrix upon initialization, filling around it, and disallowing the constrained squares from the random swaps. Data analysis showed that the algorithm performed best when the constraint was placed far from the diagonal. This makes sense since the constraint then influences fewer magic square sums. Chart C shows the average attempts required as a function of the constraint position for a level three constraint on a degree 10 square. For the constrained case where position is not specified therefore, the matrix was placed on the center of any edge.

Chart A

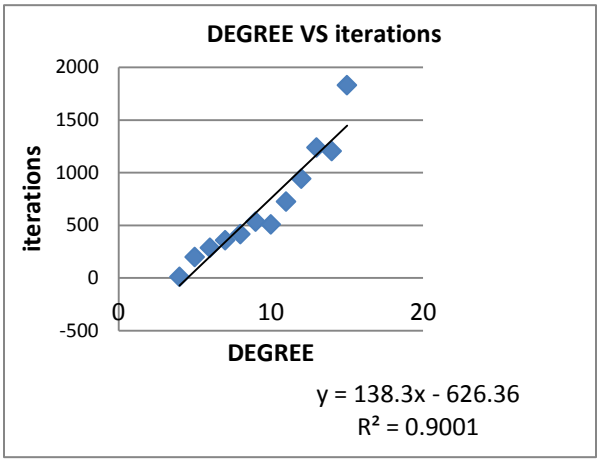

Chart B

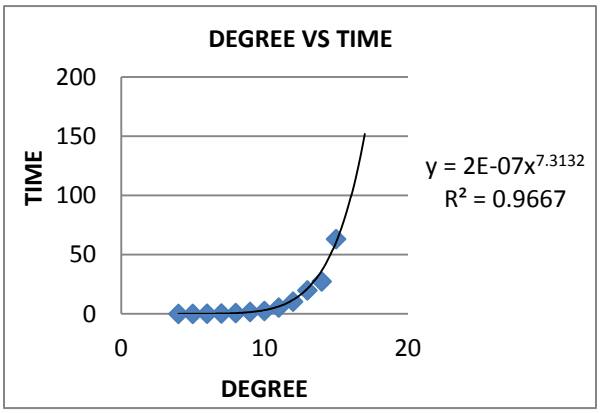

Chart C