`psychic_archer`: a solver for the Traveling Salesman Problem
CS 5600

January 3, 2014

Parker Michaelson

Introduction

The Traveling Salesman Problem (TSP) is a classic problem in computer science with many immediate theoretical and practical applications. In its original statement, it takes the form of a traveling salesman attempting to find the shortest route through a group of cities which visits each city exactly once *and* returns to the city in which the salesman started. More formally the solution to the TSP is the cheapest Hamiltonian cycle through a complete graph, with each node having a defined "cost" in-between itself and all other nodes.

Using classical (non-AI) methods, the TSP is an NP-hard problem, with a solution time of $\mathrm{O}(n!)$. While AI methods — such as the ones used in `psychic_archer` — are incapable of guaranteeing a worst case bound better than $\mathrm{O}(n!)$, AI techniques can improve the solution time in the average case of TSP. For instance, a traditional search must examine every possible Hamiltonian on the problem graph, an AI technique may be able to ignore or postpone the examination of some cycles, knowing that it is unlikely or impossible for said cycles to be the solution to the given TSP.

`psychic_archer` contains the implementation of four algorithms for solving the TSP, one classical algorithm and three AI algorithms. This report also includes an overview of the algorithms and experimental results showing the effect of problem size on the solution time for each algorithm.

Algorithm Overview

`psychic_archer` implements four algorithms for solving the TSP:

1. Search A is a traditional search through the entire TSP search space.
2. Search B is a branch and bound variant of search A. As the search progresses it remembers the best solution found so far. The algorithm will stop expanding a path should its length exceed the current best cost.
3. Search C is a variant of search B with the addition of greedy order expansion. When expanding a path, this search does so by first expanding the path to include the closest node, then expanding the path with the second closest node and so on. The idea behind this search is that the distance to the next node on the path is correlated with the total cost of finishing the Hamiltonian through that node. By trying closer nodes first, the algorithm attempts to find near-optimal solutions near the start of its run, allowing the branch and bound property of the search to rule out some cycles earlier on in the search.
4. Search D is also a variant of search B, and is similar to search C in that it uses a heuristic in an attempt to rule out many solutions early in the search. The heuristic for search D, however, is based on the assumption that the distance of the next node in the path from the *initial* node is correlated with the cost of finishing the cycle through that node. Thus search D tries to first expand to the node closest to the starting point of the path.

Methodology

All experiments used in this report were performed on an eight-core Intel i7 CPU, running under `cPython 3.3.3` and the Archlinux distribution of GNU/Linux.

Experiments to gather data on algorithm run time were performed for problem sizes n in-between seven and fourteen nodes. For each value of n, ten random node sets of size n were drawn from the file `tsp_225.txt` *via* the GNU `sed` and `shuf` programs, with each algorithm being tested once on each set. The "best" solutions to the 225 node problem contained in `tsp_225.txt` were the result of allowing each algorithm to run for one hour on the problem.

The range of experimental parameters for the Problem Size *vs.* Solution Time experiments were set in the `bash` shell script `report/timeplot/gen_times.sh`, and the data was gathered by running this script. Graphs were then generated by running the script `report/timeplot/plot_times.sh`. Similarly, the "best" solution experiments were run by executing the script `report/pathplot/gen_paths.sh`, while paths were ploted by simply running `gnuplot` with the command file `report/pathplot/plot.plot`.

Results

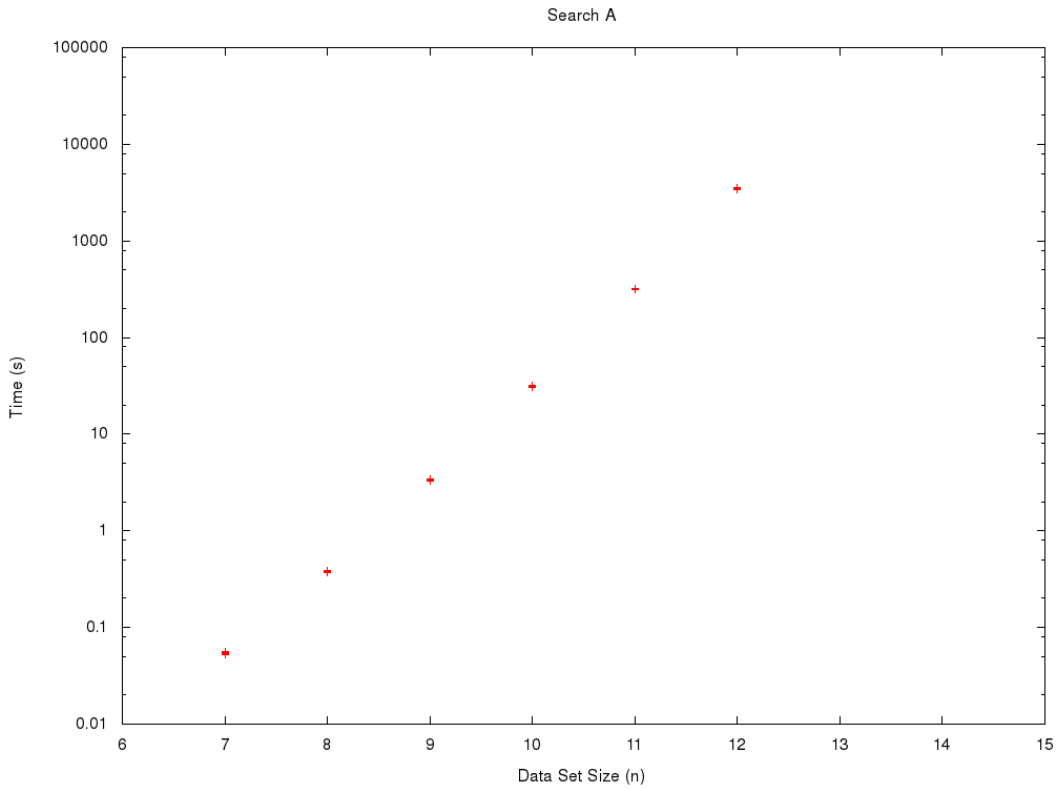

Figure 1: Search Time *vs* Problem Size, Search A

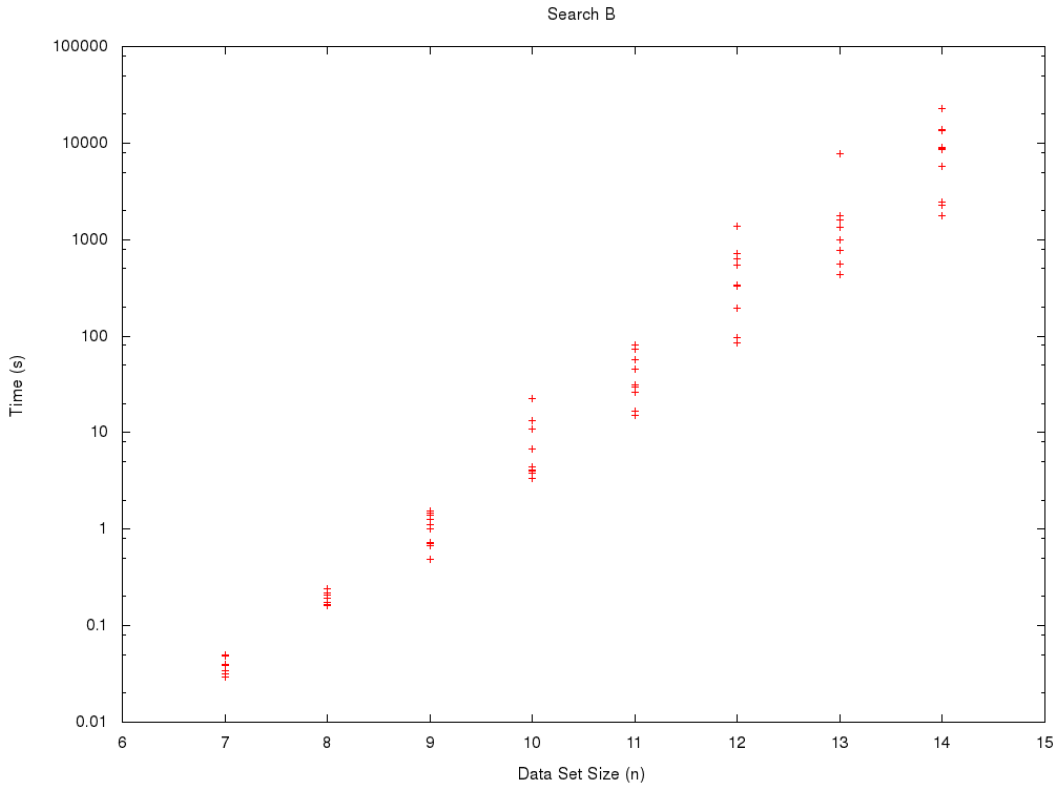

Figure 2: Search Time *vs* Problem Size, Search B

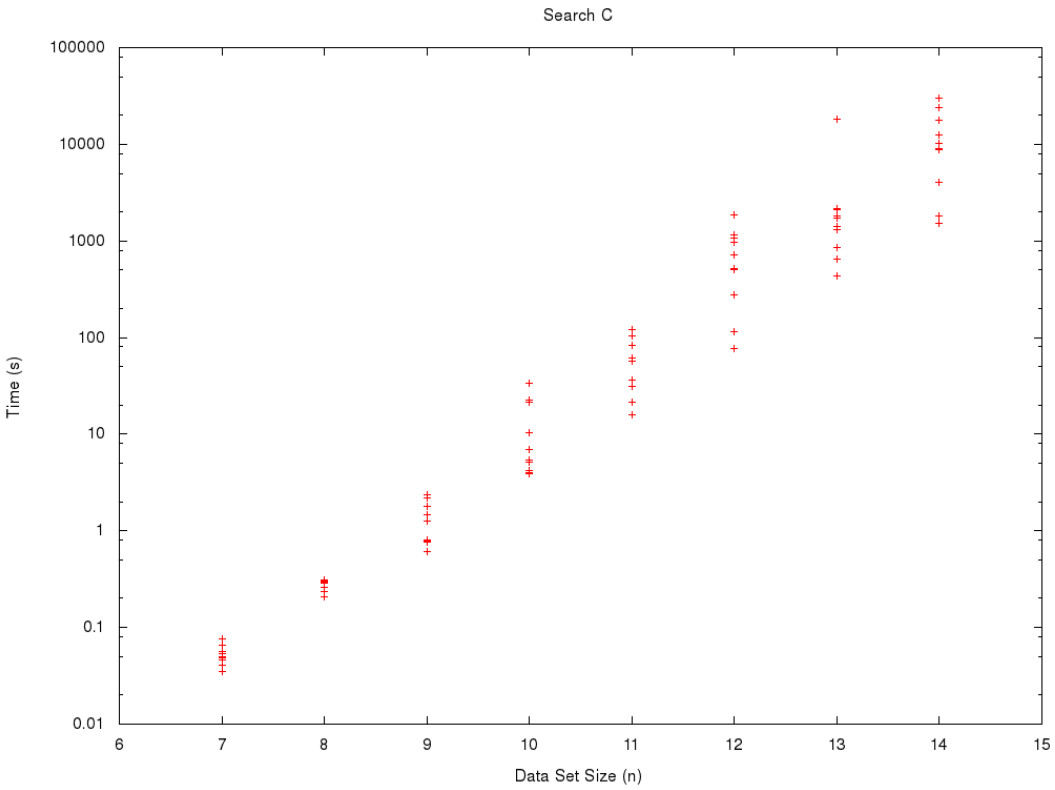

Figure 3: Search Time *vs* Problem Size, Search C

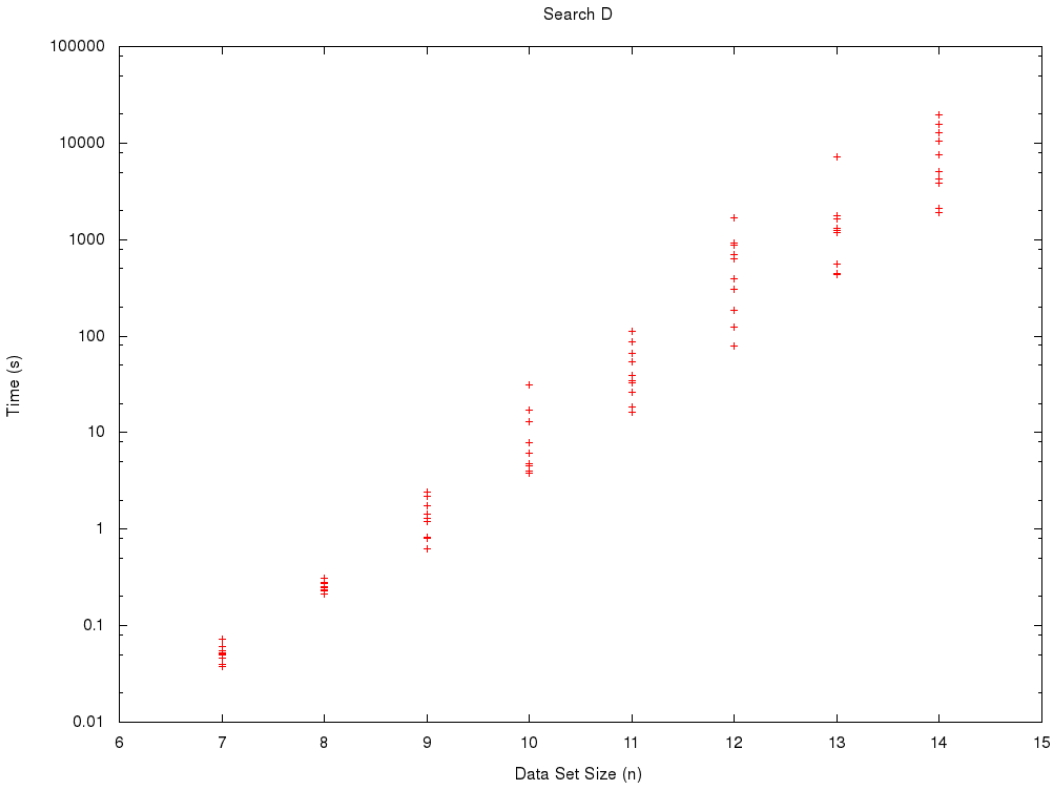

Figure 4: Search Time *vs* Problem Size, Search D

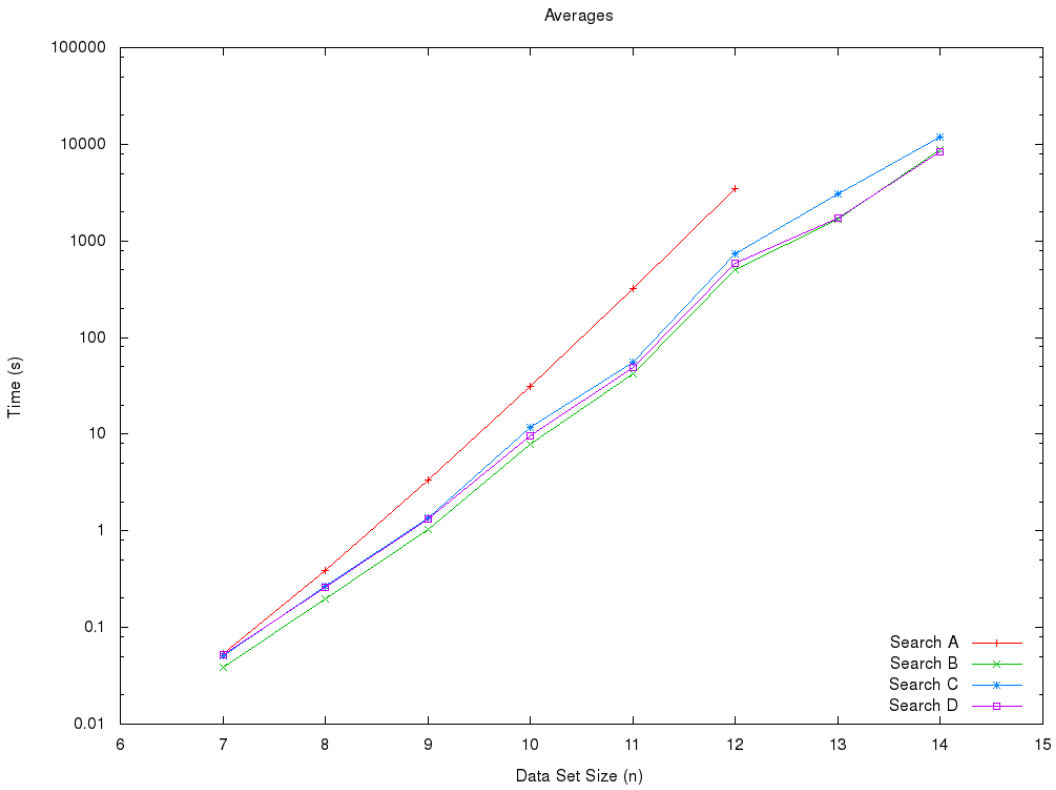

Figure 5: Search Time *vs* Problem Size, averages for all searches.

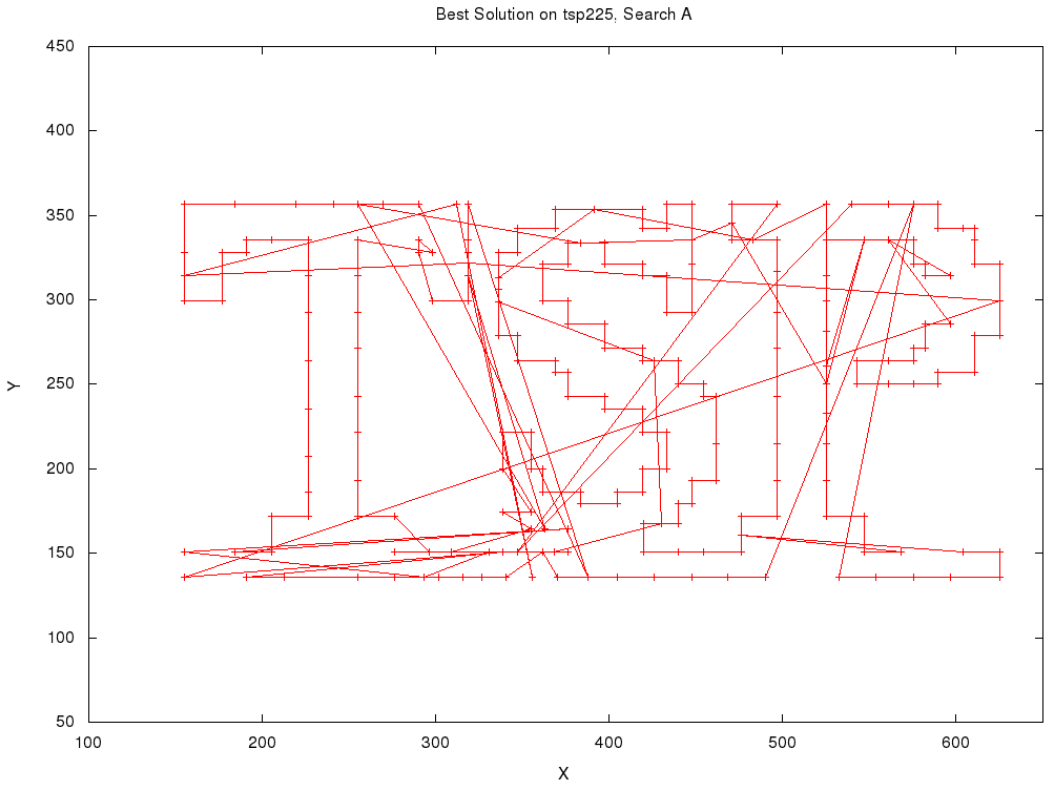

Figure 6: Best Solution on `tsp225.txt` found in one hour, Search A

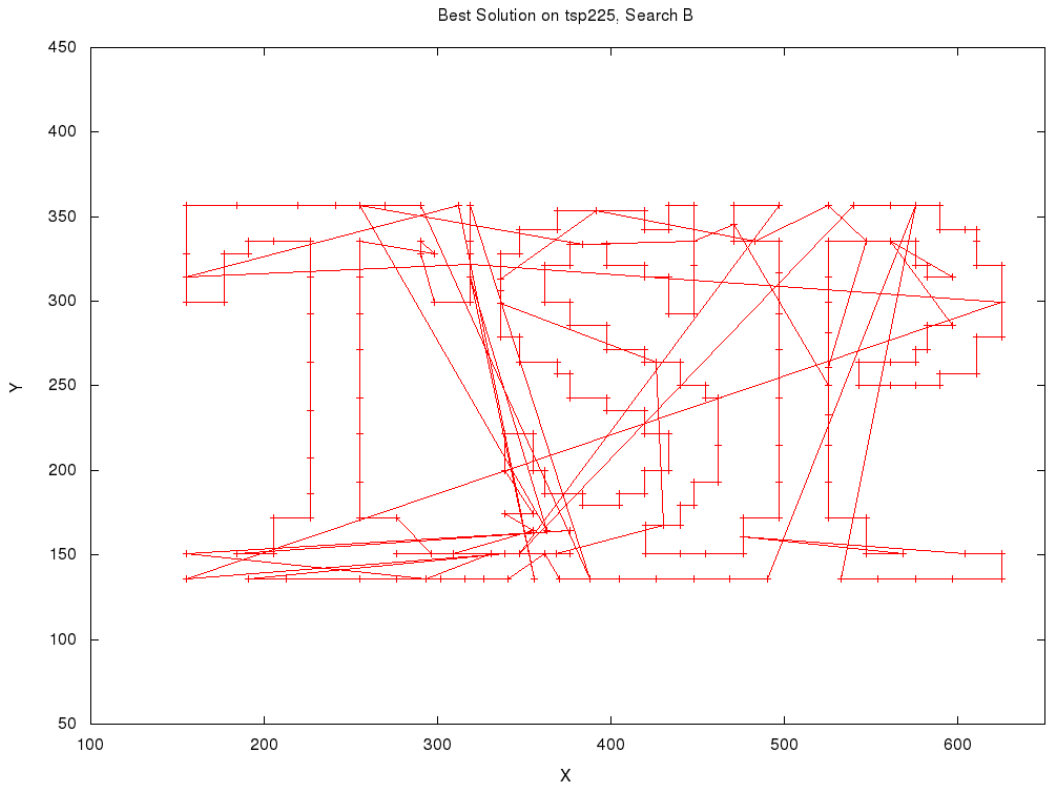

Figure 7: Best Solution on `tsp225.txt` found in one hour, Search B

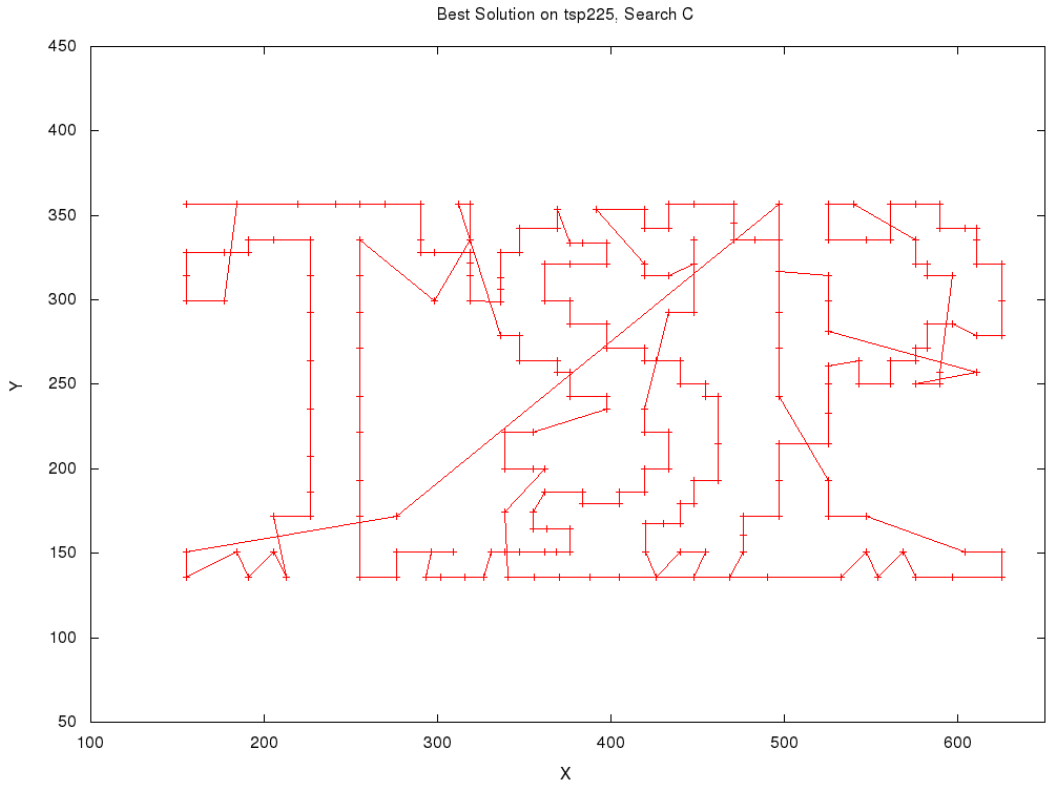

Figure 8: Best Solution on `tsp225.txt` found in one hour, Search C

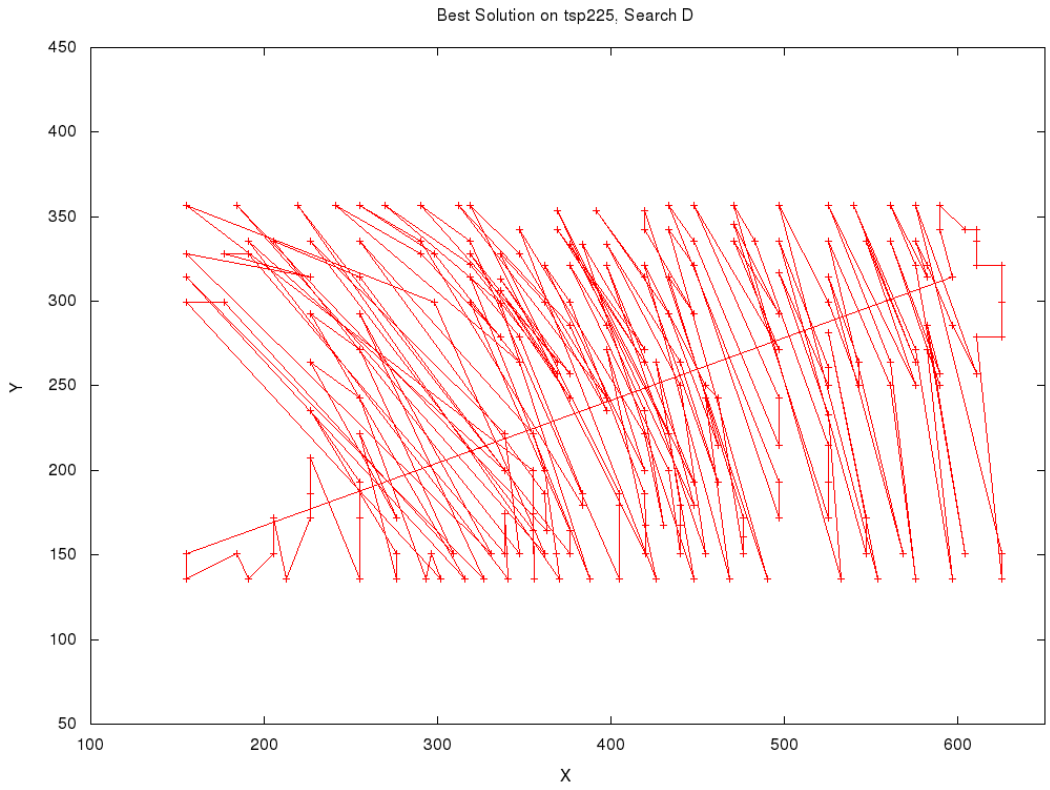

Figure 9: Best Solution on `tsp225.txt` found in one hour, Search D