CMPSC 240A Project Progress Report

Ulrik Lyng Haugen von der Freide Sagen Jr, Jakob Haug Oftebro, Eirik Aasved Holst

For our project in Parallel Computing we decided to parallelize a genetic algorithm for the traveling salesman project. It did not take long before we found some bumps in the road. Our first apparent problem was to find a way to check the validity of our solution. We've worked with a genetic algorithm for solving the NxN puzzle in CS 165A, Artificial Intelligence. For that problem we had an easy way to check our solution; if the puzzle is solved we have a solution. The travelling salesman is NP-Hard, and thus it does not have a way to check its solution in polynomial time, in fact the only way to check if the solution is correct is by solving it completely.

We have a few possible workarounds for this. First of all we will use a brute-force algorithm to find the optimal solutions for graphs up to a given n. This will mainly be important for testing as we will like our algorithm to run on problems way larger than what a brute force method ever will solve. Secondly we will try to find TSP-graphs/problems that are already solved, either on the internet or by asking Professors at UCSB. When we have correct solutions we can run our algorithm many times and find how often it actually solves the problem, how quick it converges to a solution(if at all), and how close it gets to the solution when the number of cities becomes large.

For experiments we use the methods already stated. We will do this on triton with possessors ranging from 1 to as much as we deem efficient for the algorithm, hopefully this will be close to a thousand. We will collect data to find an estimate for our worst case solution, average case solution and best case solution, for a various amount of cities. The best algorithms in the world are apparently able to get as close as 1% on graphs with cities in the millions. We hope to be able to get something similar with cities in the thousands, even though this might be an optimistic goal. We will also collect data to see how different amounts of genetic mutations (and time spent) will affect the expectation of our solution.

Our programming language of choice is Java. We have downloaded an MPI extension that allows us to run programs in parallel. We've been able to run this extension on both CSil and on Triton, so we're fully committed to complete the project in java.

We have already made a somewhat working sequential program to approximate the TSP algorithm with genetic computations. We still have some tuning to do, but after we're done with that the major task will be to get it to work on several processors. Our algorithm takes a graph, creates a bunch of random guesses at the shortest path (chromosomes). It will then take most of these guesses and evolve them by changing some cities in the path. The algorithm will always discard the chromosomes that have the worst fitness functions (for this problem we’re just using the length).

Our next goal will be to complete our sequential code and then make it program parallel. We’re going to do this by sending different parts of a population to the different processors. These processors will then perform mutations and evolve their population-part for a given amount of iterations. After that they will all send their best chromosomes (shortest solutions) back to the main node where the chromosomes will be compared. We will then repeat this process several times, a number we have not yet decided on. We will probably see how different ways of implementing this either speeds up or slows down our algorithm and the correctness of the solution it provides.