### Introduction

#### Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a type of optimization problem where we have a set of N cities, with some given distances between each of the cities. The problem is optimized by finding the shortest tour that traverses every city exactly once, but starting and ending at the same city (implied by the “tour”). The TSP solution can be organized as a list (array) where each index represents the order in which you visit the city, with the value in the list being the city itself. The search space S is a set of feasible solutions to our TSP problem, and in our case would be the set of possible permutation of cities. The neighborhood of a given potential solution s is defined by the movement, which yields a neighbor, and for the TSP is defined as a swap of two cities (one of the available permutations). The optimization of a TSP problem would thus be minimizing the total distance of the tour, given by the equation [EQUATION]. To optimize this TSP problem, we have chosen to use the Simulated Annealing metaheuristic.

### Methodology

#### Simulated Annealing – Outlined are the steps to the metaheuristic

Initial Configuration – The initial configuration of Simulated annealing is a sequence of cities in a random order (but still starting and ending on the same city, implied by “tour”). This can be implemented by taking the list of cities and randomly choosing 3 values x, i, j, where x is the number of swaps, and i and j are the indexes to be swapped. However, the starting point and ending point should be the same city. The resulting list from this implementation is a potential solution s in the search space S.

Initial Temperature – The initial temperature is calculated by computing the change in energy (denoted by X). The change in energy is computed by taking a sample of 100 movements (a permutation of two cities) and then by filling in the equation given by [EQUATION]. This can be implemented by taking an initial solution s (given by the previous implementation) and making a single swap of two indexes (which is a movement), resulting in the new potential solution s\_new. The energy of both solutions s and s\_new can be calculated using the equation referenced above, and the change in energy can be described by subtracting s\_new – s. This process is repeated 100 times, and the average can be computed by taking the sum of all the changes in energy, and dividing by 100. The choice of the initial temperature is important because we want this temperature to be high enough that we can explore the search space adequately, but we also want to initialize the temperature such that the starting acceptance/rejection probability for movements that could degrade our fitness function to be 50%.

Elementary Configuration Update – At this step, we randomly transform the solution by making a movement (permutation of two cities). Note that this movement is accepted or rejected based on the condition of the acceptance/rejection rule. This can be implemented similar to the initial configuration. The movement is defined by swapping two index locations, and those two indexes are chosen by randomly choosing 2 values, i,j, where i and j are the index values we are swapping.

Acceptance/Rejection Rule – The probability of accepting a movement is given by the equation [EQUATION]. Where we accept the swap always if the change in energy will improve the overall energy of the system (change in energy is negative). We will accept with a probability of [EQUATION] if the change in energy does not improve the overall energy of the system (change in energy is positive). The change of energy is defined by taking the energy of the neighbor solution and subtracting the energy of the current solution. The implementation is simply an application of the formula.

Equilibrium Conditions – Equilibrium is a state that is reached when a certain number of acceptances were made in the given iteration, or a total number of attempts of acceptance have been made. Once equilibrium has been reached, it is the indication that we need to lower the temperature (by a small amount) in order to continue the SA process. This can be implemented by choosing a number of acceptances and a number of total attempts, and establishing a count for each value. Once the count surpasses the number, proceed to the next iteration.

Temperature Reduction – Temperature should be lowered slowly, since quick temperature drops will make it more difficult to converge to the global minimum in favor of converging to local minimums, thus causing the original starting solution to have an effect on whether or not the algorithm converges to the global minimum, which would be bad for optimization. This implementation is simply an application of the formula, reducing the temperature by 10%.

“Freezing” Conditions – Freezing is the stop condition of the algorithm. Once temperatures have been lowered a given amount of times (3 in our case) and no improvement to the energy function has been made, the algorithm is considered to have reached the optimum. This can be implemented by checking the energy fitness at each iteration, and keeping track (in a list) the energy fitness of each iteration. If the energy fitness does not improve (meaning energy fitness of previous iterations are smaller than or equal to the current energy fitness iteration), we should end the optimization algorithm, and the solution that remains is the optimized solution.

#### Greedy Algorithm

The greedy algorithm, which is used as a baseline of comparison for this assignment, is the algorithm that takes the shortest path available at each timestep. Thus, given the neighborhood of potential movements, this algorithm always takes the neighbor with the lowest distance. We would expect that this algorithm be a good baseline since it naively takes the shortest possible path at each timestep and is thus deterministic, allowing us to compare our results.

### Results

#### Report Solutions for SA on problems attached to TP (cities and cities2)

#### Report Solutions for SA compared to a Greedy Search (Ran 10 times)

#### Graph for the optimal solution to each of the two problems (cities and cities2), for both SA and Greedy Search (Four graphs total).

#### Solve 5 randomly generated TSP problems of size 50, 60, 80, and 100. Report the MEAN, and STANDARD DEVIATION of the EXECUTION TIMES and LENGTHS OF PATHS for SA and the Greedy Search.

### Discussion

#### Discuss the results in Simulated Annealing compared to baseline Greedy Algorithm.

#### Discuss Parallel Tempering with replica number M = {2,5,20}