1 Stochastic Local Search

Problem introduction

In this section, we describe our application of stochastic local search to the traveling salesman problem, in which N cities must all be visited, without repeats, in a circuit such that the traveling distance is minimal. We also assume each city can be visited from any city other than itself. In this setting, we define the state space as the set of all such possible routes, of which there are $(N-1)!$. The objective function for this application is simply the cost, or distance, of the full route (state). We will use 4 different actions to randomly determine a new state: swapping two cities, moving a city to a new random position, inverting a sub route, and swapping the position of a sub route. We then consider the 'neighbors' of a state to be any new state that results from one of these simply transitions.

Simulated annealing (SA) is a type of stochastic local search useful for approximating the global minimum of a function. Each iteration, a new state is generated from the previous and if the cost of this new state is less then we store this new state and proceed. However, this algorithm can avoid becoming trapped in local minima by sometimes allowing worse state transitions to be accepted with some variable probability. This probability is primarily dependent on the algorithm's current *temperature*; the higher the temperature the more likely we are to accept 'poor' choices. The performance of SA depends largely on the choice of a temperature function (how temperature will decrease every iteration), the initial temperature, and the kinds of state transitions we implement.

Key features

- **Language and data structure:** Our implementation of simulated annealing is written in Python where the states are of type list of the form

  ```
  [0, 1, 2, 3, 0]
  ```

  where the above represents the path

$$\begin{tikzcd} 0 \arrow[r] & 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 0 \end{tikzcd}$$

- **Temperature initialization:** At the start of every search, generate a set of sample transitions and recursively find a T such that the acceptance probability is within a margin of error of our desired initial acceptance probability. Procedure taken from Walid, 2003[1]
- **State changes:** There are 4 different ways a path can be changed here: Swap two cities, Insert city, Swap subroute, Invert path between. Each iteration, one action is chosen uniformly at random to generate a new path. We consider the set of a state's neighbors to be all possible results from any 1 of these operations.
- **Complexity**
  - *space complexity:* $O(1)$ since we only need to reserve memory for the distance matrix, 3 paths (new, current, and best) and scalars.
  - *time complexity:* $O(n^4)$ [2]

[1] Computing the Initial temperature of Simulated Annealing, Walid 2003
[2] Optimization by Simulated Annealing: A Time-Complexity Analysis, Sasaki 1987

- **Temperature function:** Every iteration, the initial temperature is scaled down by a factor of

$$\alpha(t) = \alpha_0^t = \left(1 - \frac{1}{\text{max\_it}^{X_0}}\right)^t \tag{1}$$

  - α_0: initial value around the order of 0.9999...
  - t: current iteration (0, 1, ..., max_it) where iteration count is used as a stopping criterion and controls runtime.
  - X_0: initial desired acceptance probability ("at the start, about how likely should it be for bad paths to be picked?")

  Then $\alpha(t)$ is a number initially very close to 1 which decreases geometrically at a rate dependant on the time we plan to run the algorithm (max_it), how desirable it is to randomly choose bad transitions (X_0), and the current run time (t).

  We calculate the probability of accepting a bad transition p as

$$p = e^{\dfrac{E}{T\alpha(t)}}$$

  where E is the difference in distance between the current path and newly generated path and T is the initial temperature. If $E < 0$ then the new path is longer than the current. This means that as $t \to$ max_it, $T\alpha(t) \to 0$ and thus $p \to 0$. The result is a high probability of jumping out of local minimums early on, but slows to 0 towards the end of run time.

Path changing actions

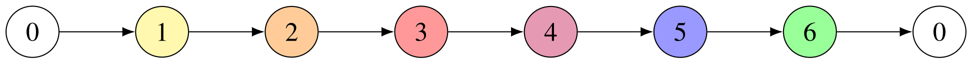

Figure 1: Initial Path

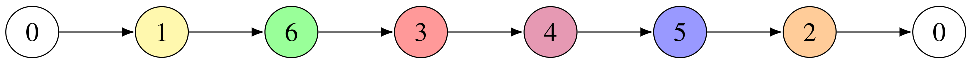

Figure 2: Switch two cities (2 & 6)

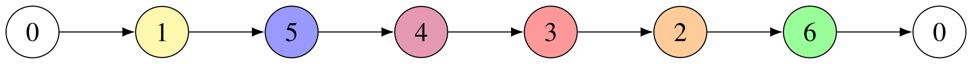

Figure 3: Invert path between two cities (2 & 6)

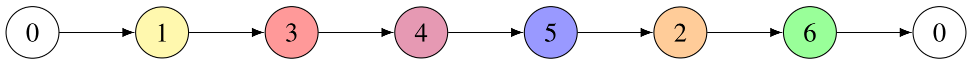

Figure 4: Insert city into random position (2 & 5)

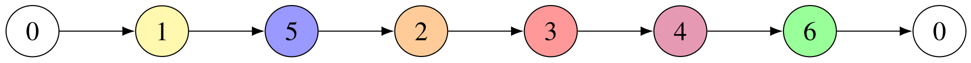

Figure 5: Swap sub route to random position (route 2-4 to 4th position)

Algorithm 1 Simulated Annealing

```
function ANNEAL(distance_mat, mat_it, $X_0$)
    $T \leftarrow$ INIT_TEMP($X_0$)                                ▷ Compute initial temperature
    current_path ← INITIAL()                                       ▷ $[0, 1, 2, ..., N, 0]$
    current_dist ← OBJECTIVE(current_path)                         ▷ Calculate path distance
    best_dist, best_path ← current_dist, current_path

    while stopping criteria not reached do                         ▷ We use max_it
        new_path ← CHOOSE_APPLY_CHANGE(current_path)
        new_dist ← OBJECTIVE(new_path)
        $E \leftarrow$ current_dist - new_dist

        if new_dist < best_dist then
            best_path, best_dist ← new_path, new_dist
        end if

        if $E > 0$ then
            current_path, current_dist ← new_path, new_dist
        else

            $p \leftarrow e^{E/T\alpha(t)}$                         ▷ where $\alpha(t)$ is as in 1

            if BINOMIAL$(1, p) = 1$ then
                current_path, current_dist ← new_path, new_dist
            end if
        end if
    end while

    return best_path, best_dist
end function
```

The above performs the simulated annealing algorithm given 3 required parameters:

- distance_mat: a symmetric $N \times N$ matrix of distances, where the $(i, j)^{th}$ element represents the distance to travel from city i to j.
- max_it: the maximum number of iterations before the algorithm terminates. Also serves to determine α_0, the initial temperature scaling value. Larger max_it will produce larger α_0 and thus the temperature stays 'hot' for longer if we are going to run for awhile.
- X_0 : the initial acceptance probability as described in Walid 2003. This value between 0 and 1 informs the initial temperature algorithm and produces starting temperatures that will, in expectation, accept bad routes $X_0 \cdot 100\%$ of the time during the actual search. We also use this value to further influence the starting α_0, such that larger X_0 leads to a higher α_0.

Several subfunctions are also defined within and contribute:

- INIT_TEMP(X_0): *described below*
- INITIAL(): Simply returns the initial path [0,1,2,...,N,0] for an N node problem.
- OBJECTIVE(path): Calculates the path distance by summing over path transition indices of the distance matrix.
- CHOOSE_APPLY_CHANGE(path): chooses with uniform probability one of the 4 switch actions and apply it to the passed path. Returns the neighbor path generated by this operation as a list.
- BINOMIAL(1,p): A Bernoulli random variable with probability of success (1) p.

Algorithm 2 Find initial temperature: problem simulation

```
1: function INIT_TEMP($X_0$)
2:     $\omega \leftarrow$ SAMPLE_SIZE(dist_mat)                    ▷ Choose appropriate sample size
3:     $\Omega \leftarrow$GEN_SAMPLE($\omega, X_0$)                  ▷ Costs of $\omega$ simulated bad transitions
4:     $T \leftarrow n > 1$                                  ▷ Initialize temperature for recursion

5:     while $|X_T - X_0| < \epsilon$ do                             ▷ for some $\epsilon > 0$
6:         $T \leftarrow T \left(\frac{\ln X_T}{\ln X_0}\right)$
7:         $X_T \leftarrow \frac{\sum_{i \in \Omega} e^{\left(-E_{a_i}/T\right)}}{\sum_{i \in \Omega} e^{\left(-E_{b_i}/T\right)}}$      ▷ $a_i, b_i$ are costs after and before $i^{th}$ trans.
8:     end while

9:     return $T$
10: end function
```

This procedure, as described in Walid 2003, calculates an initial temperature capable of providing a smooth temperature decline tailored to specific distance matrices. Since different problem instances may have very different settings (number of nodes, average city distance, variance of distances) the difference in path costs can vary greatly. For this reason, temperature is not a one-size-fits-all and we need a robust way of initializing this important variable.

To find this temperature, the above pseudo code describes a simulation process where we generate ω poor transitions (new distance longer than old distance) and record the difference in costs before and after in Ω through GEN_SAMPLE(ω, X_0). We then recursively find our initial temperature T by scaling down according to the log ratio of current acceptance probability X_T and desired probability X_0. The further off X_T is with the current T the more we adjust until we reach a specified margin of error ϵ. We typically choose a sample size ω of max_it$/10$, such that the first 10% of the runtime has about an acceptance probability of X_0.