COS314 Assignment 1 Report

Experimental setup

The experimental setup for both programs were more or less identical.

- The only difference is how “adjacent states are defined
  - ILS: States are adjacent if they differ by two consecutive campuses that are swapped around
  - SA: States are adjacent if they differ by any two campuses that are swapped around
- Both programs were written in C++ and both were compiled and executed on a Linux system.
- The cost function for both algorithms is the total cost of traversing the circuit given by the solution.
- The search space for both programs was represented as a two dimensional array with each index corresponding to a particular campus, and the index at array[i][j] corresponding to the cost of travelling between campuses i and j.
- The broad outline for both experiments can be described by the following pseudo code:

```
avgTimes = [0] * 40
avgCosts = [0] * 40
bestCosts = [Int.MAX] * 40
for maxSearchIterations in range(1, 40):
    repeat 1000:
        avgTimes[maxSearchIterations]  -= now()
        solution = searchAlgorithm(maxSearchIterations)
        avgTimes[maxSearchIterations]  += now()
        avgCosts[maxSearchIterations] += cost(solution)
        if (cost(solution) < bestCosts[maxSearchIterations]):
            bestCosts[maxSearchIterations] = cost(solution)

    avgCosts[maxSearchIterations] = avgCosts[maxSearchIterations] / 1000
    avgTimes[maxSearchIterations]  = avgTimes[maxSearchIterations]  / 1000
```

ILS Configuration

- Initial solution generated randomly
- Local search algorithm:
  - Greedy, the search algorithm iterates over consecutive pairs of campuses and swaps them if it would improve the cost of the solution (almost like bubble sort)
- Main loop:

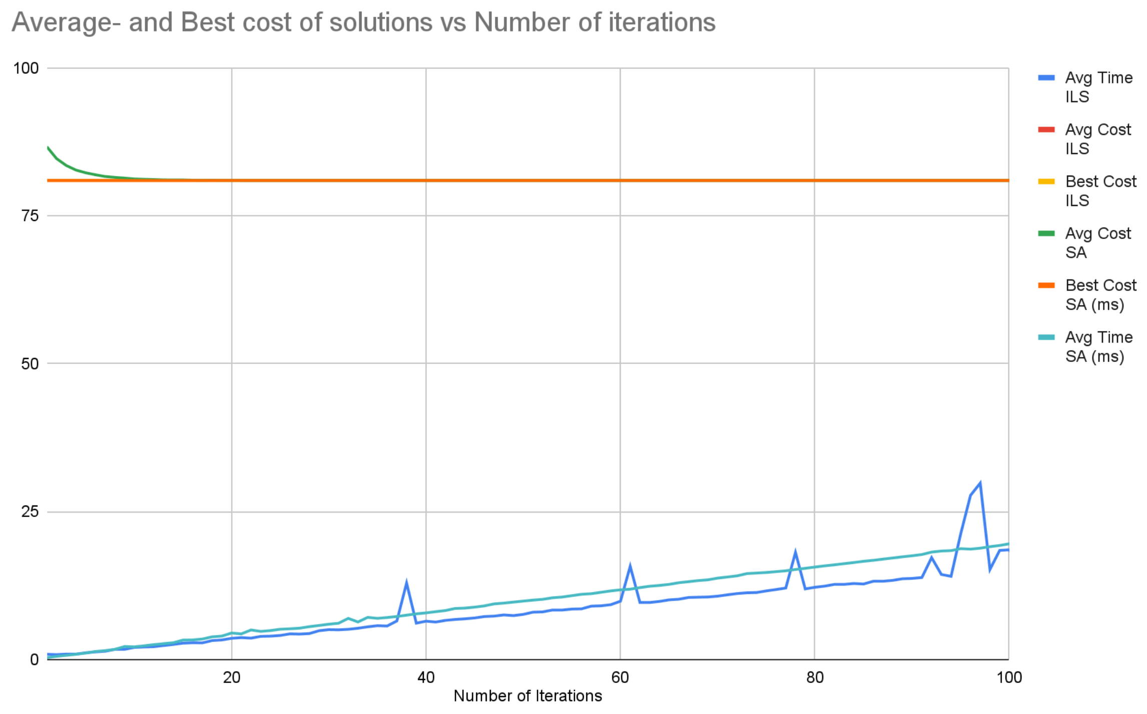
    - Perturb current best solution by swapping two random, non-adjacent campuses
    - Perform local search on perturbed solution
    - If new solution better than current best, make it the best solution
    - Repeat loop until 50 consecutive iterations have yielded no improvement

Simulated Annealing Setup

- Initial solution generated randomly
- Initial temperature set to 1000
- Cooling schedule is `Temp = Temp0 / log (iterationCount + 1)`
- Perturbation/neighbouring state generation:
    - Generate N amount of **unique** neighbours by randomly swapping any two campuses
        - N can be anything between 1 and $\binom{n}{k}$. With n being the number of campuses that can be rearranged (4; Hillcrest, Groenkloof, Prinsof, Mamelodi), and k being 2, because we can only swap two campuses at a time, we get the range (1,6). I arbitrarily decided to set N to 3.
- Main loop:
    - Set current solution to best solution
    - Generate neighbours of current solution
    - Select neighbour with best cost
    - Set current solution to neighbour solution if
        - rand(0,1] < exp((cost(current solution) - cost(best neighbour) / temp)
    - If current solution is better than the best solution, set it to the best solution
    - Repeat loop for 50 iterations

Results

Problem Set	ILS	SA
Best Solution	Hatfield, Mamelodi, Prinsof, Groenkloof, Hillcrest, Hatfield	Hatfield, Hillcrest, Groenkloof, Prinsof, Mamelodi, Hatfield
Obj Function Val	81	81
Runtime	8.43683 ms	10.0891
Avg Obj Func	81	81



The results given by the graph above do not seem particularly exciting.

Discussion

The graph reveals the following:

- Both algorithms can find the optimal solution even with only one iteration
- ILS has a slightly shorter runtime
- SA has a higher average cost when the number of iterations is lower

Best Solution

The search space is incredibly simple. It is a bi-directional, connected graph with 5 vertices. Even a simple DFS algorithm can solve it almost immediately. It is not inconceivable that both algorithms manage to find the optimal solution at least once in all the times that the algorithm executions are repeated.

Runtime

ILS has a lower runtime than SA. The reason for this is almost certainly due to the difference between how the algorithms exploit their solutions.
ILS takes a much simpler approach than SA. ILS perturbs the solution by just randomly generating two numbers that aren't adjacent or equal to each other, and then swapping those indices in the solution. ILS then performs a local search that is essentially just a bubble sort.
SA must essentially make 3 perturbations, ensure they are all unique, and then it must iterate over each of them to find the best neighbour, before replacing the current/best

solution. SA sounds simpler, but it involves more loop constructs and stack usage than the ILS implementation.

However, SA will begin to outperform ILS in this area as the search space becomes larger. SA runtime is only limited by number of iterations, whereas ILS is limited in its **consecutive number of iterations without improvement**, and as such its time complexity scales linearly with the number of iterations **and** the size of the search space.

Average Cost

The difference between the average costs for SA and ILS is due to the differences in how they explore and exploit the search space.
ILS explores and exploits in equal measure. It explores when it perturbs, and then immediately exploits the perturbed solution by performing a local search.
SA explores when it generates neighbours, and it exploits when it replaces the current solution with one of its neighbours. This is the reason why SA seems to underperform when there are fewer iterations; SA generates candidate solutions purely randomly and is much more likely to accept a worse solution during the initial stages of iteration, whereas ILS will always optimise its perturbed solution and it will never accept a solution that has a higher cost.

- This propensity for exploration can be mitigated to some extent by modifying the initial temperature and the cooling schedule of the SA algorithm.
- Another method of limiting exploration is to generate more neighbours. SA explores by generating random neighbours and accepting slightly worse solutions to escape local minima/maxima. The more neighbours that are generated, the better the probability of finding a better solution– in which case the solution is exploited and a worse solution will not be accepted to explore. But one must take care to not generate too many neighbours, as there is a lot of overhead involved in generating unique neighbours.
- If the temperature is reduced to a certain extent, and the number of neighbours is increased, the SA algorithm essentially just functions like a BFS algorithm.

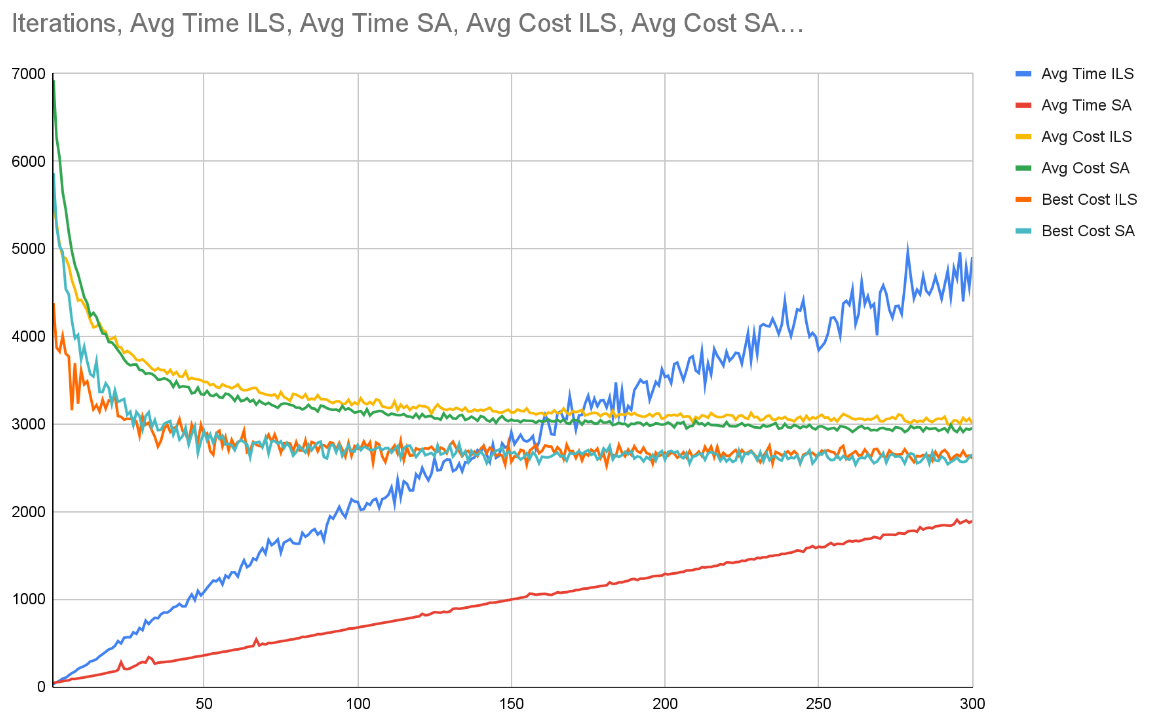
However, as the number of iterations increases and the temperature of SA is allowed to cool, it will become less likely to accept worse solutions and become more exploitative. This is clearly illustrated by the convergence of the average costs for ILS and SA.

ILS and SA with a larger Search Space

The small search space seems to be the reason why ILS outperforms SA in both runtime and average cost. As such, I wanted to see how they perform on a larger search space. I've created a search space with 29 points on a plane, with the cartesian distance between them being used for calculating cost.

```
double vectors[29][2] = {
    {519.5, 378},   {371.5, 351},   {210, 472.5},   {405.5, 193},
    {405.5, 453.5}, {485.5, 465.5}, {656.5, 68.5}, {614.5, 341.5},
    {417.5, 519},   {393.5, 251.5}, {434, 39.5},    {526.5, 531},
    {468.5, 261},   {337.5, 83},    {405.5, 139},   {558, 220},
    {259.5, 51.5},  {321, 127},     {490, 153.5},   {359.5, 275.5},
    {429.5, 382.5}, {330.5, 25},    {706, 232},     {551, 307},
    {558, 107.5},   {330.5, 484.5}, {605, 285},     {551, 419.5},
    {294, 439}};
```

Results



Once again, ILS starts off with a lower average cost, but both algorithms seem to converge as the number of iterations increases and SA surpasses ILS after a while, but the difference between their average cost seems consistent.

SA starts with an abysmally low value for its best costs, but it soon converges with ILS as it did in the case of average costs.

The largest difference is that the runtime values have reversed. The runtime of ILS is much more closely correlated to the size of the search space, because more iterations are required to find improved solutions and ILS extends its execution time whenever it finds an improved solution.

Conclusion

ILS and SA are more or less equivalent in the application of TSP based on a bi-directional, connected graph. ILS has worse time complexity due to both its focussed exploitation of every candidate solution, as well as its execution time being constrained by a period of no improvement. SA has better time complexity, due to its highly constraining iteration limit. ILS performs better on smaller data sets due to its focussed exploitation. SA performs better on larger data sets due to its extensive exploration phase, which is also the reason it performs sub-optimally on smaller datasets.