A Comparative Analysis of ILP and Approximation Algorithms for the 2-Dimensional Euclidean TSP

Ben Rosenberg

2025-05-13

Introduction

In this report, we will compare several algorithms for obtaining solutions or approximations to the 2-Dimensional Euclidean TSP. Specifically, we explore the following solution methods:

- Christofides' algorithm, a polynomial time method of obtaining an approximate solution to the metric TSP with an upper bound on the ratio of the given tour to the true optimum value of the TSP of 1.5
- Simulated annealing, a meta-heuristic that can be combined with various methods to improve a solution and obtain iteratively better approximations
- Integer Linear Programming, a technique involving modeling a problem as a series of linear equations and/or inequalities that will yield an optimal solution at the cost of potentially exponential runtime

Implementations of each approach are compared and recommendations are provided based on the problem-solving situation. We find that there is a tradeoff between runtime and accuracy, with simulated annealing doing well as a compromise between these two metrics.

Background

Let $G = (V, E)$ be a complete graph, and let $v \in \mathbb{R}^2, \forall v \in V$. Define $w : E \to \mathbb{R}$ as the weight of each edge in E, corresponding to the L_2 distance between v_1 and v_2.

A **Hamiltonian circuit** in G is a sequence of nodes $C = (v_1, v_2, \ldots, v_n, v_1)$ where:

- $v \in V, \forall v \in C$
- $|C| = |V| + 1$
- $v_1 \neq v_2 \neq \cdots \neq v_n$

Since G is a complete graph, our definition does not need to describe any restrictions on the relationship between consecutive nodes in C, because every possible pair of consecutive nodes corresponds to an edge in E.

Define $W(C)$ to be the sum of all edge weights $w(v_n, v_1) + \sum_{i=1}^{n} w(v_i, v_{i+1})$ implied by the order of nodes in $C = (v_1, v_2, \ldots, v_n, v_1)$.

We will explore the Traveling Salesman Problem (TSP) on G, which is known as the Euclidean TSP given the constraints on G described above.

Let $H(G)$ denote the set of Hamiltonian circuits in G. The TSP is the problem of finding a Hamiltonian circuit of minimal total edge weight, C^*:

$$C^* = \underset{H(G)}{\operatorname{argmin}}(C \mapsto W(C))$$

Algorithmic approaches

This problem has been widely studied and is known to be NP-hard. Because of this, many approximation heuristics have been developed, aiming to obtain better runtime with minimal impact $W(C^*) - W(C)$ of the solution's approximation on the objective value compared to the optimum.

Christofides' algorithm

Christofides' algorithm, originally given in Christofides (1976), is a method of obtaining an approximate solution to the metric TSP (of which the Euclidean TSP is one manifestation), starting with a minimum spanning tree (MST) in the graph. The algorithm follows these steps (enumerated in Genova and Williamson (2017)) on a graph G:

- Find a minimum spanning tree T in G
- Identify vertices with odd degree in T
- Find a minimum-weight perfect matching M of these odd-degree vertices
- Combine edges of T and M to form a graph $T \cup M$
  - This set is connected because T is conected, and satisfies the TSP constraint that all nodes must have even degree, because we

added one edge to all vertices with odd degree
- Find an Eulerian circuit (that is, a circuit that touches all edges) in $T \cup M$
- Create a TSP tour by shortcutting the Eulerian circuit

This algorithm was introduced in the above-cited 1976 paper by Nicos Christofides, in which he also described the runtime as $O(n^3)$. For the TSP, an $O(n^3)$ runtime is not bad, especially when compared with the exponential worst-case runtime we will see later for the ILP formulation.

Also in the original paper, Christofides described the approximation produced by the algorithm as being upper-bounded by 1.5 times the optimal TSP solution.

Simulated annealing

Simulated annealing is a meta-heuristic that can be applied to a variety of problems, as long as two objective values can be compared to determine which is better, and there is a way to perturb a solution to obtain one which is still valid but may have a different objective value.

The heuristic method works as follows:

- We start with a randomized solution, and some temperature T
- Iteratively explore the neighborhood of the current solution, perturbing the solution minorly
- After the solution is perturbed, decide whether to accept the new solution or not based on T and some acceptance function that takes the difference in objective value as an input
- Gradually decrease T over time, so that the heuristic closes in on some local optimum

2-opt

We can explore the neighborhood of a TSP solution using a technique known as **2-opt**, which works as follows:

- Select two edges to delete
- Add two edges to reconnect the tour

An example of 2-opt in action is provided in Figure 1.

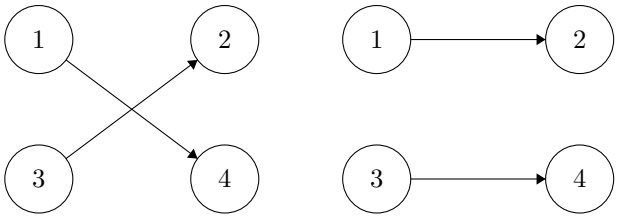

Figure 1: Before (left) and after (right) the application of 2-opt

Each iteration of 2-opt requires $O(n^2)$ time, as it compares each pair of edges in the tour, and the tour has n edges.

A similar heuristic is 3-opt, which also performs a comparison-dependent transformation, but with 3 edges instead of 2. However, 3-opt requires $O(n^3)$ time per iteration, so may not be as efficient e.g. in simulated annealing.

So, with 2-opt, we can apply simulated annealing to the TSP as follows:

- Choose some random permutation $(v_1, \ldots, v_n)$ of V, and append v_1, to get an initial solution
- Explore the neighborhood with 2-opt
- Compare solution to incumbent objective and decide whether to accept or reject the transformation based on T

ILP formulation

The last method of obtaining a TSP solution we will discuss is the ILP formulation. An ILP, or integer linear program, is a model for a problem that exclusively uses linear inequalities and/or equalities (and constrains some variables to be integers).

There are several different ILP formulations for the TSP; we will explore the Dantzig-Fulkerson-Johnson formulation (Dantzig, Fulkerson, and Johnson 1954), which is widely known.

The ILP formulation for the TSP begins with these constraints:

- C1: There must be exactly one edge that leaves each node
- C2: There must be exactly one edge that enters each node

Denote $\{1, \ldots, n\}$ by $[n]$. If we let $x_{ij} \in \{0, 1\}$ be the variables that denote whether the edge from i to j is used in the resulting tour, then we can represent the above two constraints, along with the objective

function, as follows:

$$\min \sum_{i\in[n]} \sum_{j\in[n]-i} w(i,j)\cdot x_{ij}, \text{ s.t.}$$

$$\text{C1}: \sum_{i:\in[n]-j} x_{ij} = 1 \qquad \forall j \in [n]$$

$$\text{C2}: \sum_{j\in[n]-i} x_{ij} = 1 \qquad \forall i \in [n]$$

However, with these constraints alone, subtours could be permitted, as in Figure 2.

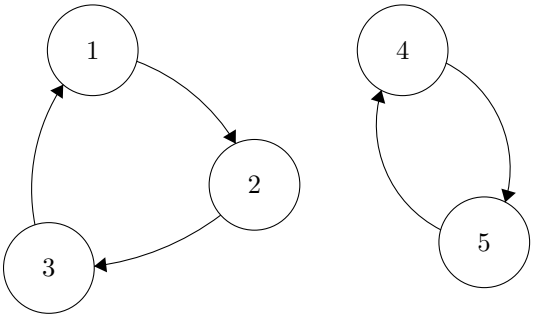

Figure 2: Subtours

To avoid subtours, we need to introduce subtour elimination constraints. In the formulation we are using these are incorporated by ensuring that no subtour has as many edges as it has nodes:

$$\sum_{i\in S} \sum_{j\in S-i} x_{ij} \leq |S| - 1 \quad \forall S \in \mathcal{P}([n]), 2 \leq |S| < n$$

Because $|\mathcal{P}(n)| = 2^n$, this formulation may require exponentially many of these subtour elimination constraints, but there exist methods to generate these constraints as needed by detecting subtours. It has been theorized (Pferschy and Staněk 2017) that for random Euclidean TSP instances, the expected number of subtour elimination constraints that are required to be generated could be polynomially bounded; however, this remains an open question.

There are various techniques to solve ILPs, but they typically have the following steps:

- Solve the linear programming (LP) relaxation of the ILP, in which the constraint that variables are integers is removed. This can usually (with some caveats - see Spielman and Teng (2001)) be done in polynomial time with the simplex method
- Use a technique such as Branch and Bound to obtain an optimal integer solution. This may require exponential time, as we may need to check each possible value on all of the n^2 variables x_{ij}, requiring 2^{n^2} such checks

In practice, the TSP ILP can be solved quickly using heuristics such as branch-and-cut (Mitchell 2002), but large instances still pose problems for even state-of-the-art solvers (for example, the TSP tour of 81,998 Korean bars found in (Cook et al. 2025) took 44 years of compute). We will discuss this limitation further in the next section.

Implementation

We now briefly describe the method of implementation of each of these algorithms in Python, which we use to roughly compare the algorithms in various metrics.

Christofides' algorithm

The algorithm of Christofides is implemented in the `networkx` package (Hagberg, Schult, and Swart 2008). The algorithm is well-studied and the implementation given in `networkx` is well-tested; furthermore, implementing the algorithm itself is outside the scope of a report of this nature.

Simulated annealing

The simulated annealing approach was implemented using 2-opt, which was described above.

Pseudocode for the implementation follows:

```
SimulatedAnnealingTour(cities)
------------------------------
current_route <- random_permutation(cities)
current_dist <- tour_dist(current_route)
best_route <- current_route
best_dist <- current_dist
temp <- initial_temp
while temp > stopping_temp do
    for iterations_per_temp iterations do
        new_route, new_dist, diff <- 2opt(
            current_route
        )
        if diff < 0 or prob_accept(diff, temp) then
            current_route <- new_route
            current_dist <- new_dist
            if current_dist < best_dist then
                best_route <- new_route
                best_dist <- new_dist
    temp <- temp * cooling_rate
return best_route, best_distance
```

The `prob_accept` function is a probabilistic acceptance function that depends on the current temperature and the difference between the old solution and new solution. A typical function to use for this probabilistic acceptance was introduced in the original paper on simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983):

$$P(\Delta E, T) = e^{-\frac{\Delta E}{T}},$$

where acceptance is governed by the comparison of a uniform $U(0,1)$ random variable with $P(\Delta E, T)$.

We chose the following initial conditions rather arbitrarily in order to get reasonable convergence times:

- `initial_temp`: 1000
- `cooling_rate`: 0.99
- `stopping_temperature`: 0.1
- `iterations_per_temp`: 100

These parameters led to a typical convergence period of under 10 seconds when running on instances up to 150 vertices in size.

ILP formulation

The ILP formulation was implemented using OR-Tools (Perron and Didier 2025). The formulation used was the above-mentioned DFJ ILP formulation.

Results

Figures 3 through 5 compare execution time, memory usage, and total distance by method. Figure 6 compares the accuracy of the approximation algorithms (Christofides, simulated annealing) against the optimal solution (the ILP formulation).

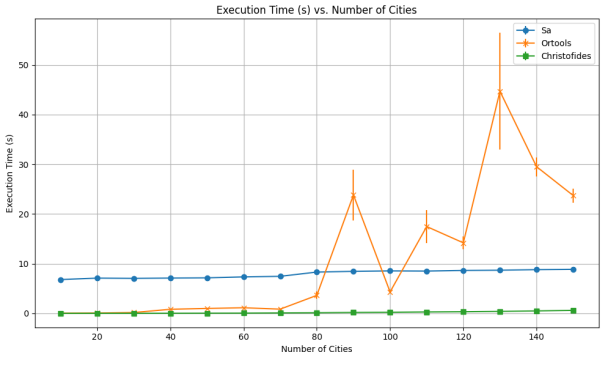

Figure 3: Execution time

These results corroborate the analysis of the solution methods given above, with some exceptions:

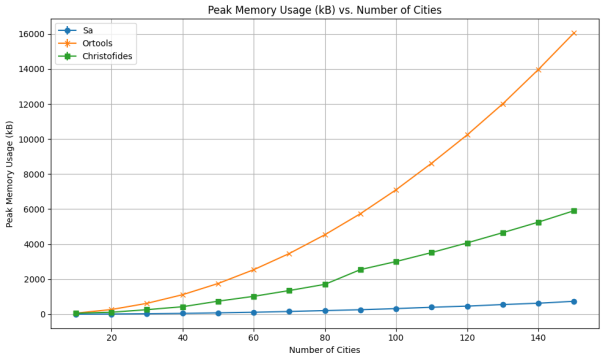

Figure 4: Memory usage

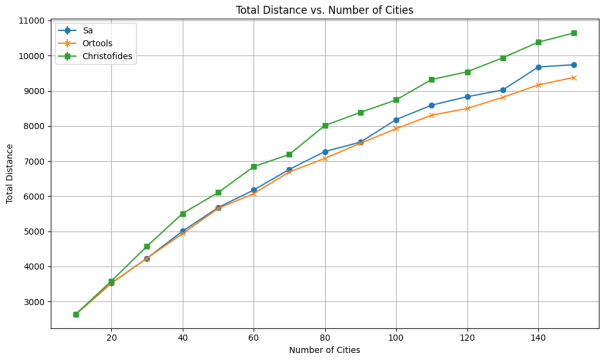

Figure 5: Total distance

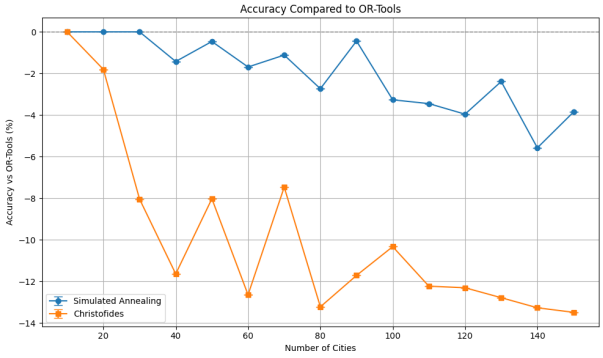

Figure 6: Accuracy

- The execution times of the ILP formulation is rather erratic for larger instances. This may be due to quirks of the implementation, or random chance (there were only 10 runs done per city size).
- In the examples we are use to compare the three algorithms, Christofides performs well within the "1.5 times optimal" bound. This is likely because the examples are smaller, and there are therefore fewer opportunities for Christofides to make poor choices. Additionally, in our implementation, we use an average of 10 runs, which is unlikely to reflect the upper bound given for a single solution.

Additional caveats on the results follow:

- The performance of simulated annealing is dependent on the parameters for initial temperature, cooling factor, and stopping temperature. Different choices of these parameters may have led to better results.
- Memory usage is heavily implementation-dependent. Christofides and the ILP formulation likely required significantly more memory than simulated annealing because they use external libraries.
- The execution time was measured using wall time, not CPU time.

Conclusions

The algorithm that should be chosen to solve the TSP depends on the use case at hand:

- Simulated annealing can take some time to converge, and its efficiency relies heavily on implementation. Even within a given implementation, the chosen constants can incorporate a tradeoff between performance and accuracy.
- Christofides' algorithm can be a decent approximation, and is guaranteed to run in $O(n^3)$.
- The ILP formulation will produce an exact solution, but may take significantly longer time when dealing with larger instances.

In general, the simulated annealing approach is recommended as a tradeoff between execution time and accuracy. The simplicity of implementation for simulated annealing is another benefit not previously mentioned - it was easily implementable without the use of an external library. For those interested in using simulated annealing but not interested in implementing it, `networkx` (Hagberg, Schult, and Swart 2008) supports a simulated annealing solution method in its `algorithms.approximation.traveling_salesman` library alongside its implementation of Christofides.

References

Christofides, Nicos. 1976. "Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem." *Operations Research Forum* 3 (1). https://doi.org/10.1007/s43069-021-00101-z.

Cook, William, Daniel Espinoza, Marcos Goycoolea, and Keld Helsgaun. 2025. "Korea81998 Computation." *Korea81998: Shortest-Possible Walking Tour to 81,998 Bars in South Korea*. University of Waterloo. https://www.math.uwaterloo.ca/tsp/korea/computation.html.

Dantzig, G., R. Fulkerson, and S. Johnson. 1954. "Solution of a Large-Scale Traveling-Salesman Problem." *Journal of the Operations Research Society of America* 2 (4): 393–410. http://www.jstor.org/stable/166695.

Genova, Kyle, and David P. Williamson. 2017. "An Experimental Evaluation of the Best-of-Many Christofides' Algorithm for the Traveling Salesman Problem." *Algorithmica* 78 (4): 1109–30. https://doi.org/10.1007/s00453-017-0293-5.

Hagberg, Aric A., Daniel A. Schult, and Pieter J. Swart. 2008. "Exploring Network Structure, Dynamics, and Function Using NetworkX." In *Proceedings of the 7th Python in Science Conference*, edited by Gaël Varoquaux, Travis Vaught, and Jarrod Millman, 11–15. Pasadena, CA USA.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi. 1983. "Optimization by Simulated Annealing." *Science* 220 (4598): 671–80. http://www.jstor.org/stable/1690046.

Mitchell, John. 2002. "Branch-and-Cut Algorithms for Combinatorial Optimization Problems1." *Handbook of Applied Optimization*, January.

Perron, Laurent, and Frédéric Didier. 2025. *CP-SAT* (version v9.9.3963). Google. https://developers.google.com/optimization/cp/cp_solver/.

Pferschy, Ulrich, and Rostislav Staněk. 2017. "Generating Subtour Elimination Constraints for the TSP from Pure Integer Solutions." *Central European Journal of Operations Research* 25 (1): 231–60. https://doi.org/10.1007/s10100-016-0437-8.

Spielman, Daniel A., and Shang-Hua Teng. 2001. "Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time." https://doi.org/10.48550/ARXIV.CS/0111050.