Travelling Salesman Problem

David Nardi
MSc in Artificial Intelligence
david.nardi@edu.unifi.it
https://github.com/david-inf/Python-TSP

June 2024

Abstract

With this work we tackle the Travelling Salesman Problem (TSP) using few known algorithms. We start with two local search approaches that swap the edges of the cycle, once we saw that the local searches get trapped in local minima we moved forward to a multi-start meta-heuristic for each local search. Simulated annealing is then tested using a routine to initialize the temperature parameter. Results show that...

Python implementation is available at this GitHub repo.

Contents

ToC Modelling the TSP

1. Ops perspective
2. Smoothing the hard constraint, the pragmatic perspective

Tryna solve the TSP

1. Local search
   - `swap`
   - `swap-rev` or `reverse`
2. Simulated annealing
3. Multi-start

Results

- Convergence analysis and other diagnostics
- Energy view using multi-start
- What's next? Simulated tempering (SA with reheating), genetics algorithms

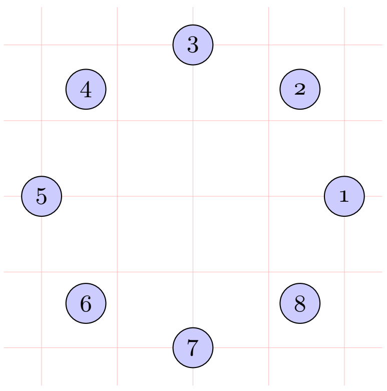
References

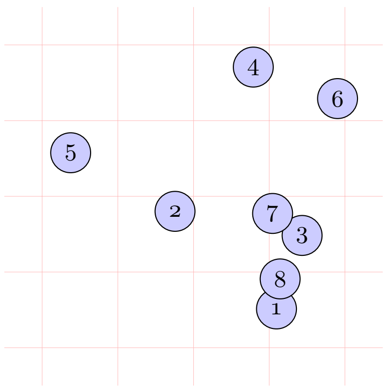
- Collega di Pisa? Quelli che trovavo su Medium? Paper trovati per caso?

1 Modelling the travelling salesman problem

Given an undirected (weighted) and complete graph $G = (V, E)$, where the set of all nodes $|V| = N$ and the set of all edges $\{i, j\}$ is $|E| = N(N-1)/2$ and their relative weight $c_{ij} > 0$. an Hamiltonian cycle is a cycle which touches every node in V once and only once, an Hamiltonian cycle might not exist.

We consider the two following layouts with all possible edges





on the left, a circular graph, for which we know that the optimal solution is a circular path; on the right each node is in a random position, so we don't know exactly the optimal solution.

We constraint the Hamiltonian cycle to start with the first node 1, so the total number of possible cycles reduces to $(N-1)!$.

1.1 The optimization perspective

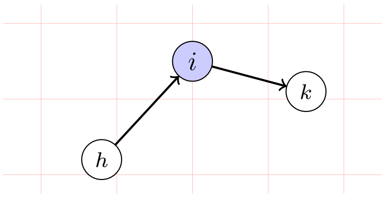
The Travelling Salesman Problem (TSP) consists in finding the *shortest Hamiltonian cycle*. The feasible set F is the set of all Hamiltonian cycles in G and the objective function $c(P)$ is the length of the Hamiltonian cycle P.

We make use of a logical variable for each edge

$$x_{ij} = \begin{cases} 1 & \text{if } i \to j \\ 0 & \text{otherwise} \end{cases}$$

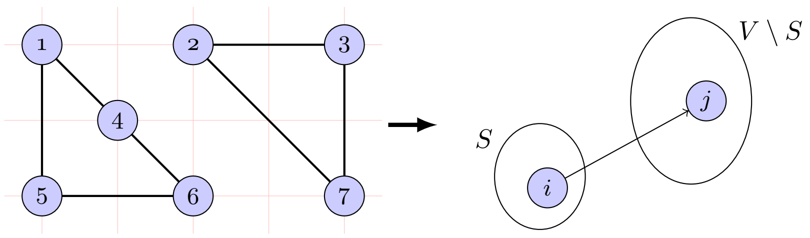
hence the objective function si the sum of all costs associated with the edges in the cycle. This is the binary constraint on variables.

Since a feasible solution is a cycle, we must impose the constraint that each node has one incoming and one outgoing edge



that is the *cycle covering* constraint.

Using only this constraint can result in a solution with sub-tours, so we add a constraint for connecting all possible sub-tours



from a subset S there must be at least an edge to the complementary set $V \setminus S$, the *connection* constraint, that is the *hard* constraint.

The resulting optimization problem will be

$$\begin{aligned} \min \quad & c(P) = \sum_{\{i,j\}\in E} c_{ij}x_{ij} \\ & \sum_{\{i,j\}\in E} x_{ij} = 2 \\ & \sum_{i\in S, j\notin S} x_{ij} \geq 1 \quad \emptyset \subset S \subset V \\ & x_{ij} \in \{0,1\} \end{aligned} \tag{1}$$

In practice, for simplicity, the cost for each edge is the distance between the two cities, so we use the distance matrix $D \in \mathbb{R}^{N \times N}$

$$D = (d_{ij}) = \begin{pmatrix} 0 & d_{12} & d_{13} & \cdots & d_{1N} \\ d_{21} & 0 & d_{23} & \cdots & d_{2N} \\ d_{31} & d_{32} & 0 & \cdots & d_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{N1} & d_{N2} & d_{N3} & \cdots & 0 \end{pmatrix}$$

that is a symmetric matrix, whether an edge is in the cycle or not, the variables x_{ij} become the elements of the adjacency matrix A of the same shape of D. The objective function will be $c(P) = A \odot D$ that we call f for simplicity; the cycle covering constraint becomes the sum over rows and columns of A that must be equal to $2N$.

What kind of algorithms?

- heuristics
- meta-heuristics

Which algorithms?

- Brute-force
  - Swap indices
  - Swap indices and reverse middle indices
  - Check all permutations
- Meta-heuristics
  - Multi-start (GRASP)
  - Simulated Annealing (SA)

1.2 Smoothing the hard constraint

1.3 Local search

The idea behind this class of algorithms is simple: given a feasible solution, there might be some other similar feasible solutions with lower objective function value. So the idea is to optimize the objective function by exploring the neighborhood of the current point in the solution space.

The local search starts with a feasible solution randomly drawn in the feasible set. A generation mechanism is then successively applied in order to find a better solution, in terms of the objective function value, by exploring the neighborhood of the current solution.

The algorithm ends when no improvement can be found, and the current solution is considered as the approximate solution of the optimization problem.

Algorithm 1: Local search framework

```
Input: $f$, $D$, $x^0$
1 $x^* \leftarrow x^0$;
2 $k \leftarrow 0$;
3 while *stopping criterion not satisfied* do
4     Generate a feasible solution $\bar{x}$ s.t. $\bar{x} \in I(x^*)$;
5     if $f(\bar{x}) < f(x^*)$ then
6         $x^* \leftarrow \bar{x}$;
7     end
8     $k \leftarrow k + 1$;
9 end
Output: $x^*$
```

If the current solution falls in a subdomain over which the objective function is convex, the algorithm remains trapped in this subdomain. In order to avoid being trapped in local minimum, it is then necessary to define a process likely to accept current feasible solutions that momentarily reduce the performance (objective function value) of the current solution.

Algorithm 2: Multi-start framework

```
Input: $f$, $D$
1 $x^*$ s.t. $f(x^*) = \infty$;
2 for $b = 1, 2, \dots, B$ do
3     Generate a starting feasible solution $x^0$;
4     Perform a local search (algorithm 1) starting from $x^0$ to obtain $x^b$;
5     if $f(x^b) < f(x^*)$ then
6         $x^* \leftarrow x^b$;
7     end
8 end
Output: $x^*$
```

1.4 Simulated annealing

Definition 1 (Metropolis acceptance criterion)**.** Let (S, f) be an instantiation of a combinatorial minimization problem, and i, j two points of the state space. The acceptance criterion for accepting solution j from the current solution i si given by the following probability:

$$\mathbb{P}(\text{accept } j) = \begin{cases} 1 & \text{if } f(j) < f(i) \\ \exp(-(f(j) - f(i))/c) & \text{otherwise} \end{cases}$$

where c is the control parameter.

Definition 2 (Transition)**.** A transition represents the replacement of the current solution i by a neighboring solution. This operation is carried out in two stages: generation and acceptance.

Algorithm 3: Simulated Annealing (SA)

```
Input: $f$, $D$, $x^0$, $T_0$, $\alpha$, $L_k$
$x^* \leftarrow x^0$;
$k \leftarrow 0$;
while stopping criterion not satisfied do
    $x^k \leftarrow x^*$;
    for $i = 1, 2, \ldots, L_k$ do
        Generate a feasible solution $\bar{x}$ s.t. $\bar{x} \in I(x^*)$;
        if $f(\bar{x}) < f(x^*)$ then
            $x^* \leftarrow \bar{x}$; // down-hill, new best solution
        end
        if $f(\bar{x}) < f(x^k)$ then
            $x^k \leftarrow \bar{x}$; // down-hill note that $f(x^k) \geq f(x^*)$
        else
            Generate a random number $r \sim U(0, 1)$;
            $\Delta E \leftarrow f(\bar{x}) - f(x^k)$; // energy gap
            if $r < \exp(-\Delta E/T_k)$ then
                $x^k \leftarrow \bar{x}$; // up-hill, lower quality solution accepted
            end
        end
    end
    $T_{k+1} \leftarrow \alpha T_k$;
    $k \leftarrow k + 1$;
end
Output: $x^*$
```