Tryna Solve the 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 heuristic 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 the local search gets trapped in local minima due to the starting solution, multi-start solves this problem; the simulated annealing takes much more computation time but ends always in a better solution. Further studies may focus on reheating procedures for the simulated annealing like the simulated tempering.

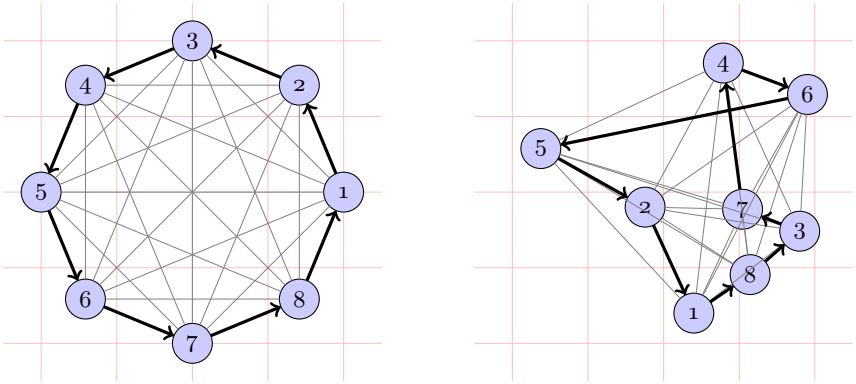
Python implementation is available at this GitHub repo.

Contents

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 cost (weight) $c_{ij} > 0$.

We consider the two following layouts (circular and random) with all possible edges and a feasible solution for each one is displayed



on the left, a circular graph, for which we know that the optimal solution is a circular path, that is 1, 2, 3, 4, 5, 6, 7 and 8 but the same reversed too since the total cost is the same; on the right each node is in a random position, so we don't know exactly the optimal solution, but a good feasible solution can be found in a polynomial time varying with the problem size.

We constraint the Hamiltonian cycle to start with the first node 1, so that the total number of possible cycles reduces to $(N-1)!$, a practical trick since we see the cycle as a sequence of all the nodes in the graph.

1.1 From an optimization perspective

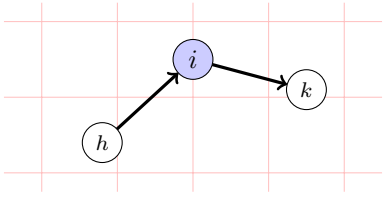
The Travelling Salesman Problem (TSP) consists in finding the *shortest Hamiltonian cycle*. The feasible set F is the set of all Hamiltonian cycles in G and the objective function $c(P)$ is the length of the Hamiltonian cycle P.

We make use of a logical variable for each edge

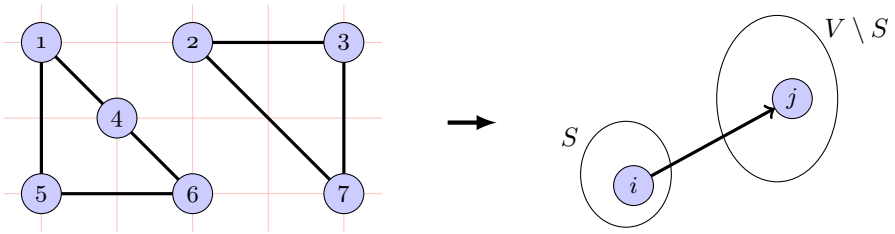
$$x_{ij} = \begin{cases} 1 & \text{if } i \to j \\ 0 & \text{otherwise} \end{cases}$$

hence the objective function is 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, the *hard* constraint of the optimization problem.

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 Euclidean distance between the two cities, so we can use the distance matrix $D \in \mathbb{R}^{N \times N}$

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

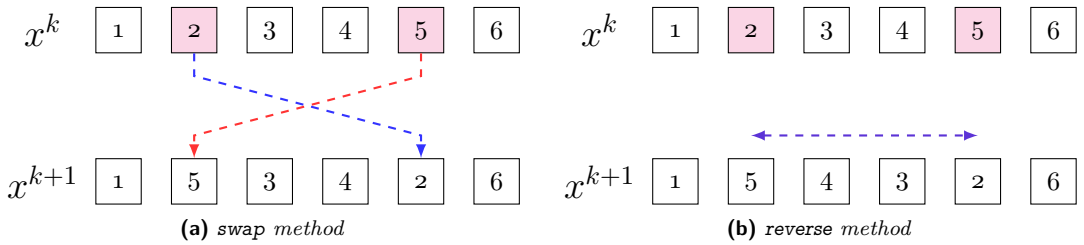
that is a symmetric matrix, whether an edge is in the cycle or not, the variables x_{ij} become the elements of the adjacency matrix A of the same shape of D. The objective function will be $c(P) = A \odot D$ that we call $f(x)$ for simplicity with x a certain solution (sequence); the cycle covering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

1.2 Smoothing the hard constraint, practical TSP

Each one of the algorithms considered starts with a starting feasible solution x^0 and then performs a perturbation on that solution in order to find a new neighbouring solution $x^k \to x^{k+1}$. We can smooth the hard constraint by keeping this solution each time and swapping nodes in the sequence so that the new solution is still an Hamiltonian cycle.

Here we have used two different methods, see figure 1 on the following page, that randomly draw two different nodes i and j given a current solution x^k:

- `swap`: given the sequence, swap positions i and j, figure 1a;
- `reverse`: given the sequence, reverse the position of each nodes between i and j inclusive, figure 1b.



(a) *swap method* **(b)** *reverse method*

Figure 1: Perturbation methods for generating new solutions, defined with permutation encoding. This methods will be used in local search and simulated annealing algorithms.

2 Heuristic algorithms

Heuristic algorithms do not guarantee the optimal solution but may give good solutions, we made use of local search and meta-heuristics in this work.

2.1 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 neighbourhood of the current best solution x^k in the *solution space*.

The local search starts with a feasible solution randomly drawn from the feasible set $x^0 \in F$. A generation mechanism is then successively applied in order to find a better solution, in terms of the objective function value $f(x^k)$, exploring the neighbourhood means perturbing the current solution, hence for this purpose we use the methods from figure 1.

The algorithm ends when a maximum number or iterations k^* has been exceeded, and the current solution is considered as the approximate optimal solution of the optimization problem.

2.2 Multi-start

Local search algorithms always ends in a local minima that are not the global optimum, this is due certainly to the perturbation method and the starting solution x^0 as well.

In order to make the local search independent from x^0, we can use a multi-start meta-heuristics which involves performing local searches starting from different initial values. Basically, we choose a number B of generations, for each one a starting solution is randomly generated, then a local search is performed; finally the best solution is chosen.

This procedure is called Greedy Randomized Adaptive Search Procedure (GRASP), for large B we can be sure to obtain the global optimum since it is part of the feasible set from which we draw the starting solutions.

2.3 Simulated annealing

Local search falls in a subdomain over which the objective function is convex, in order to avoid being trapped in a local minima, it is necessary to define a process likely to accept current feasible solutions that momentarily reduce the objective function value.

Simulated Annealing (SA) can realize this idea, the acceptance of new solution is controlled by a control parameter, the *temperature*, in such way the algorithm can consider past informations

about the optimization process: once the algorithms ends in a good solution, similar solutions can be quite near the current one.

The simulated annealing uses the following parameters:

- T_k: *temperature*, starting from a initial value T_0, this parameter drives the search of the global optimum. It must be iteratively tuned in order to reach a high *acceptance rate* on initial (outer) iterations like $\chi(x) \geq 0.8$ that is the ratio, in each inner iterations, between the number of accepted solution (*transition*) and proposed ones.
- α: *cooling rate* for the temperature parameter according to geometric cooling $T_{k+1}\alpha T_k$, the parameter will slowly tend to zero;
- L_k: *length of the Markov Chain* for which $T_k = \text{cost}$, it is the number of inner iterations for each k outer iteration, on which new solutions are generated.

Inside each transition a new feasible solution x^t is generated as in the local search through methods from figure 1 on the preceding page; from the statistical mechanics perspective, these perturbations allows to work in the canonical ensemble: the energy E_t (the objective function) is free to fluctuate but the number of particles (nodes in the graph) remains fixed.

Once a solution is generated the algorithm checks if there is an improvement over the current best solution x^* (the sequence of $f(x^*)$ is constrained to be decreasing); then the Metropolis acceptance criterion (Monte Carlo importance sampling) is applied using the energy gap $\Delta E_t = f(x^t) - f(x^k)$, this rule checks if there is an improvement over the current transition x^k, the criterion is as follows

$$\mathbb{P}(\text{accept } x^t) = \begin{cases} 1 & \text{if } f(x^t) < f(x^k), \text{ i.e. } \Delta E_t < 0 \\ \exp(-\Delta E_t / T_k) & \text{otherwise, i.e. } \Delta E_t \geq 0 \end{cases}$$

the rule accepts the new solution based on the Boltzmann distribution which is used for the transition probability, this criterion allows to accept up-hill moves that increase the objective function value $f(x^k)$, so the sequence $\{f(x^k)\}_k$ will fluctuate.

Algorithm 1: Local search framework

Input: f, D, x^0

$x^* \leftarrow x^0$;
$k \leftarrow 0$;
while *stopping criterion not satisfied* **do**
    Generate a feasible solution x^k, see figure 1;
    if $f(x^k) < f(x^*)$ **then**
        $x^* \leftarrow x^k$;
    end
    $k \leftarrow k + 1$;
end

Output: x^*

Algorithm 2: Multi-start framework

Input: f, D
x^* s.t. $f(x^*) = \infty$;
for $b = 1, 2, \ldots, B$ **do**
  Generate a starting feasible solution x^0;
  Local search (algorithm 1) from x^0 to $\hat{x}^b$;
  if $f(\hat{x}^b) < f(x^*)$ **then**
    $x^* \leftarrow \hat{x}^b$;
  end
end
Output: x^*

Algorithm 3: Simulated Annealing (SA)

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

3 Results

In order to solve the problem, we firstly explore the *solution space* using the `multi-start` metaheuristic on different TSP instances. The considered problems have different sizes: $N = 30$ and 50 for circular and random layouts.

Exploring the solution space means to empirically find the *energy landscape*, that is the distribution of each $f(x^*)$ of the problem. The multi-start method is applied with local search and simulated annealing as base algorithms.

We keep track of each final objective function value and each solution because two different solutions may have the same f (in circular layout the circular path clockwise and counter-clockwise), so the *ratio* between the number of unique local minima and starting solutions can be computed.

Figure 2 on the next page shows these results. Increasing the number of local search iterations k for a given problem instance reduces the ratio and we can see that the mode of the distribution shifts towards left, however as the problem size increases, the number of sharp local minima on which the algorithms may fall increases as well.

A greater size of the problem must see a larger number of iterations for the local search, we see that the mode shifts left and the distribution has a normal shape, this means that the most frequent local minimum isn't the best one since there are other values on the left. The random layout has more local minima on which the local search may fall.

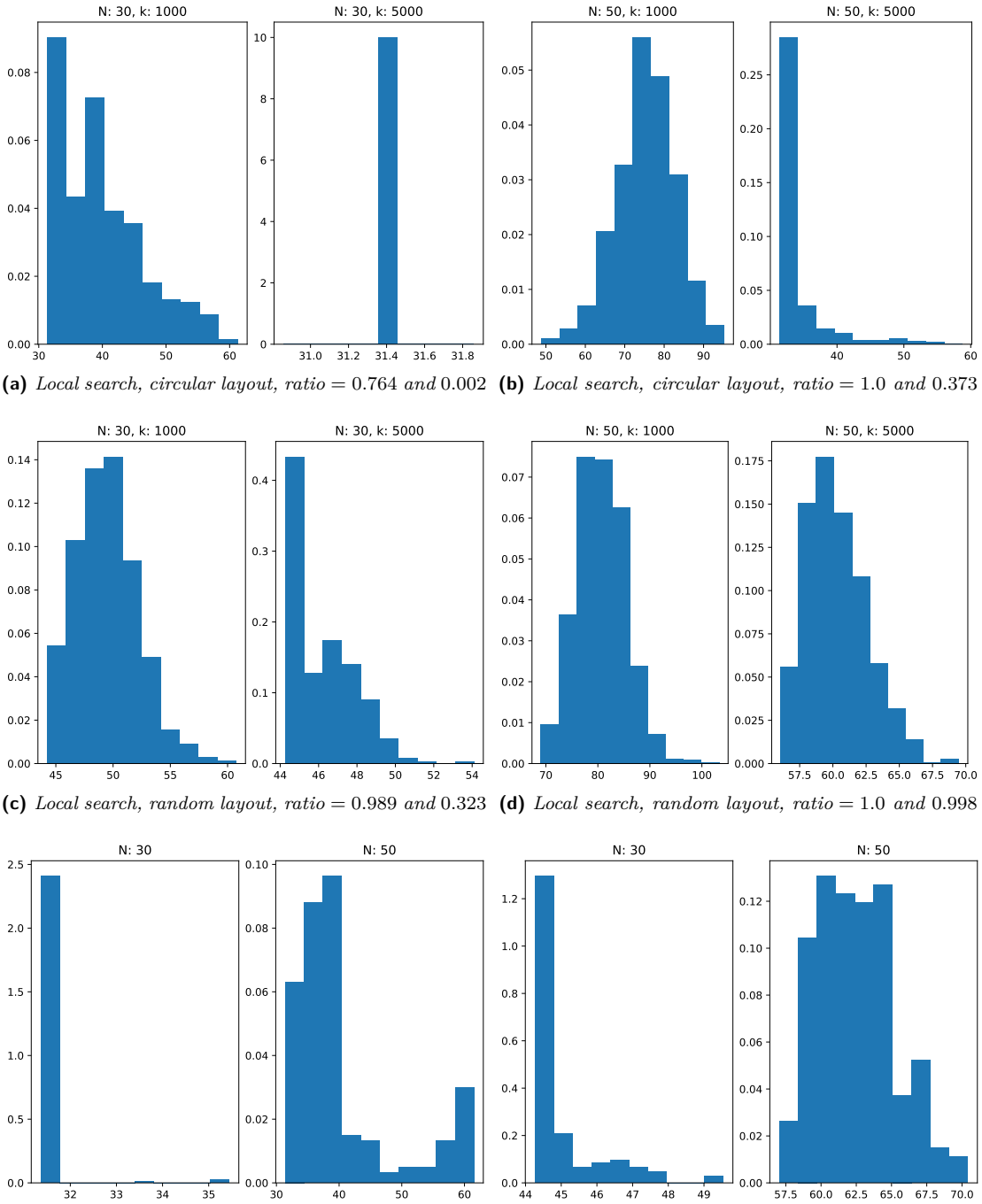
When the size of the problem is quite large $N = 50$ the local search needs a large number of iterations, so we can increase further k or move to another algorithm. However, local search will eventually fall in a local minima, but the multi-start approach can ease this situation since the local search is dependent on the starting solution and we provide a pool of x^0.

While the simulated annealing can escape sharp local minima, it is still dependent by the starting solution, so a multi-start approach can help this algorithm too. For both algorithms we see that the random layout objective function has much more local minima and even the SA might fall in one of them if not carefully tuned, i.e. increase k and most important L_k.

Now we can move forward to solve the problem instances with the previous methods, figure 3 displays the diagnostic for the best local search obtained through the multi-start and figure 4 on page 10 for the simulated annealing.

In both diagnostic we can see that the solutions with lower objective function value have no crossing edges. Results show that local search easily gets trapped in local minima, especially the `swap` method, while `reverse` proves to be a better choice, however both need far more iterations to reach $f(x^*)$ closer to simulated annealing.

In the simulated annealing diagnostic best and last accepted objective function values performance are displayed, we can see that when the acceptance rate is about $\chi(x) \approx 0.5$ the $f(x^k)$ series has a negative trend and so converges with $f(x^*)$ while χ goes to zero, the algorithm stops when no more solution are accepted. When the problem size increases, not much the outer iterations as the length of the Markov chain needs to be increased.

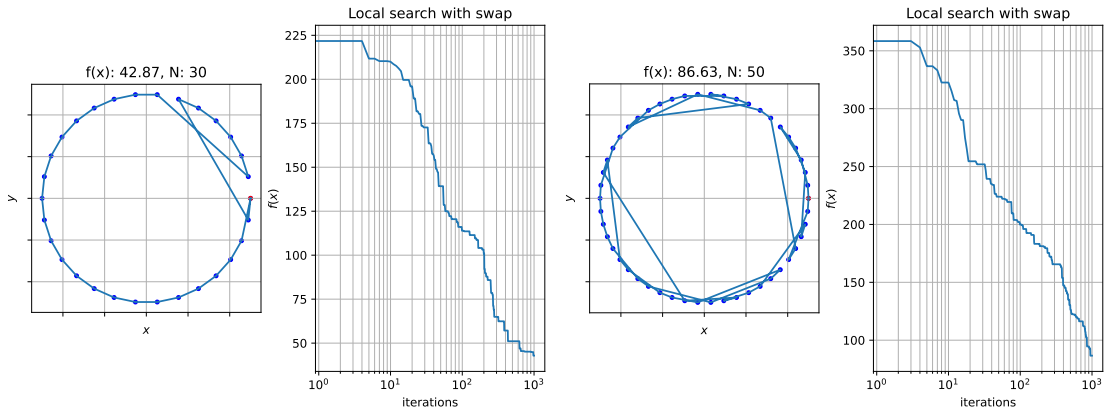


(a) *Local search, circular layout, ratio* $= 0.764$ *and* 0.002 **(b)** *Local search, circular layout, ratio* $= 1.0$ *and* 0.373

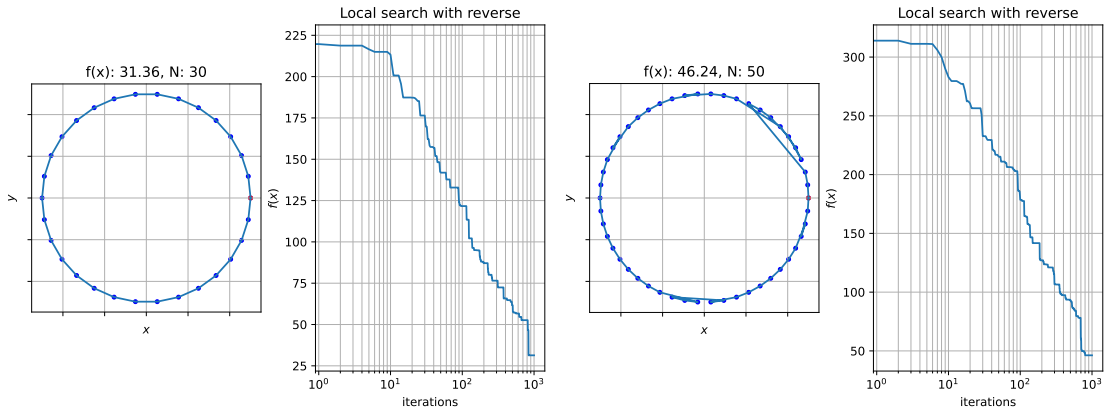
(c) *Local search, random layout, ratio* $= 0.989$ *and* 0.323 **(d)** *Local search, random layout, ratio* $= 1.0$ *and* 0.998

(e) *Simulated annealing, circular layout, ratio* $= 0.025$ *and* 0.985 **(f)** *Simulated annealing, random layout, ratio* $= 0.465$ *and* 1.0

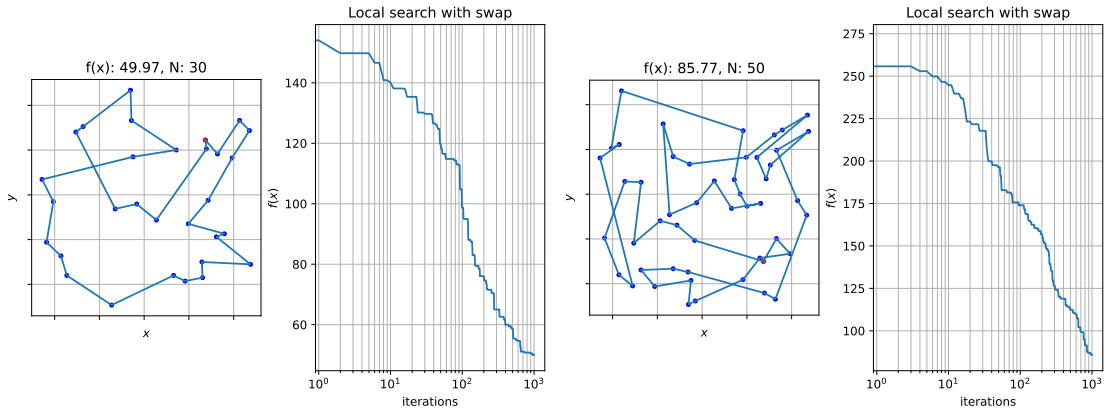
Figure 2: Local search with `reverse` method (k iterations, 1000 starting solutions) and SA ($k = 500$, $L_k = 200$, $\alpha = 0.995$, 200 starting solutions) energy landscapes for $N = 30$ and 50 instances



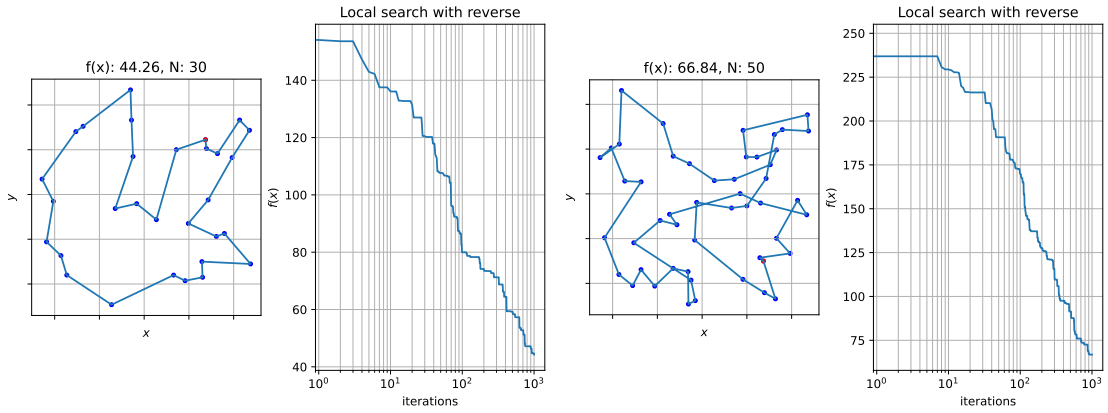
(a) *swap method on circular layout* **(b)** *swap method on circular layout*



(c) *reverse method on circular layout* **(d)** *reverse method on circular layout*

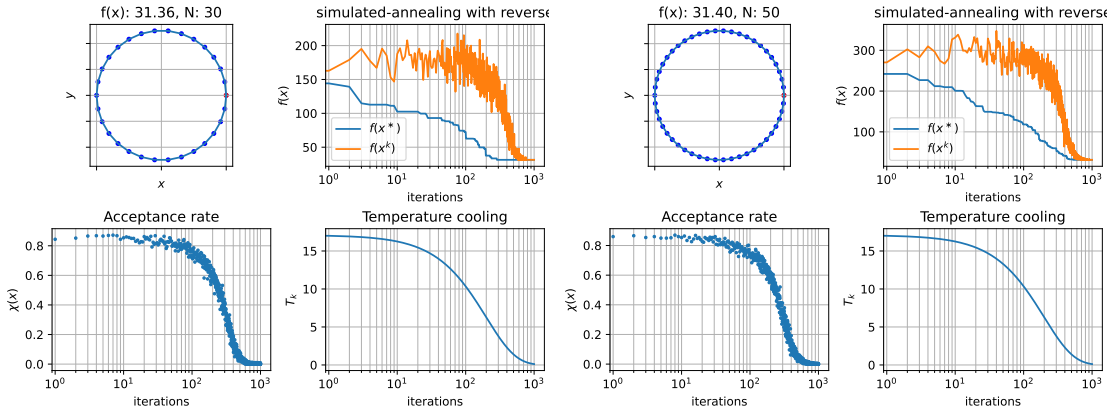


(e) *swap method on circular layout* **(f)** *swap method on circular layout*

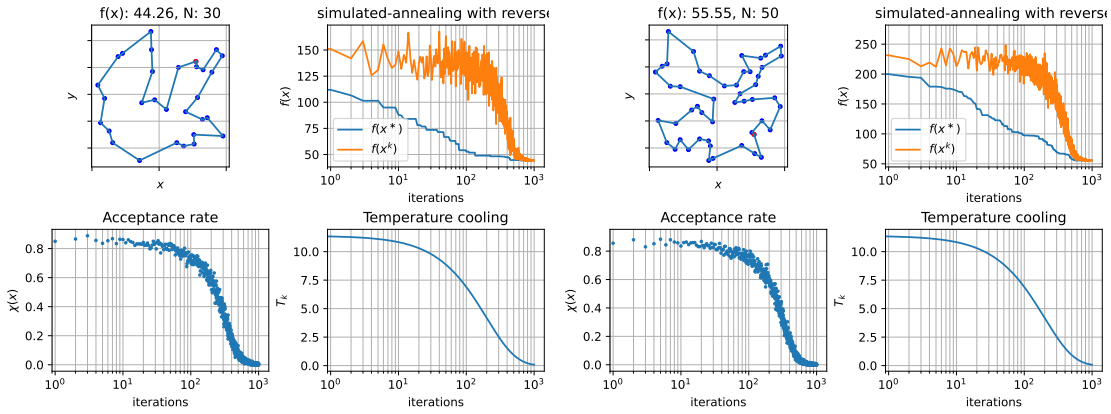


(g) *reverse method on circular layout* **(h)** *reverse method on circular layout*

Figure 3: Local search algorithms ($k = 1000$) performance on problem instances. Used a multi-start procedure for selecting the best solver from a pool of 1000 starting solutions



(a) *Circular layout* **(b)** *Circular layout*



(c) *Random layout* **(d)** *Random layout*

Figure 4: Simulated annealing ($k = 1000$, $L_k = 500$, $\alpha = 0.995$) performance on problem instances. Best solution selected through a multi-start method from 10 different starting solutions

References

[1] G. Bigi, A. Frangioni, G. Gallo, S. Pallottino and M. G. Sculettà, *Appunti di ricerca operativa*, 2024. [Online]. Available: https://commalab.di.unipi.it/wp-content/uploads/2024/05/v1.0.0-240424.pdf.

[2] D. Delahaye, S. Chaimatanan and M. Mongeau, 'Simulated annealing: From basics to applications,' in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds. Cham: Springer International Publishing, 2019, pp. 1–35, ISBN: 978-3-319-91086-4. DOI: 10.1007/978-3-319-91086-4_1. [Online]. Available: https://doi.org/10.1007/978-3-319-91086-4_1.