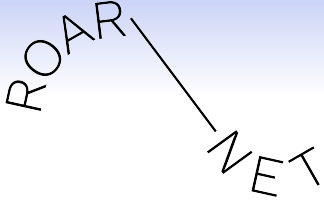


Local Search

1st ROAR-NET Training School

Francesca Da Ros

cost
EUROPEAN COOPERATION
IN SCIENCE & TECHNOLOGY

Funded by
the European Union

Contents

Introduction

Introduction

ROAR NET

- Algorithm paradigm
  - Hill Climbing, Tabu Search, Simulated Annealing, etc. + *variations*.
- Selective techniques
  - Exploration of the search space composed of complete solutions.
  - vs. Constructive: Start from an empty solution and build the complete one step by step.
- Non-exact techniques (also called approximate techniques)
  - No guarantee of finding the optimal solution nor proving optimality.
  - vs. Exact: Eventually find the proven optimal solution.

⚠️ Still, it depends on the problem structure

A simple & old idea

- *Navigate* the search space by stepping from one state (i.e., solution) to one of its *neighbors*
  - **Neighborhood**: Set of states you can reach by applying *local change* to it.
  - **Local change**: Given a state, modify a few of its variables
- First literature dates back 50-60 years ago [Sörensen et al., 2018].

Why still popular?

🏆 Delivered good results in real-world application and competitions.

- Some examples:
  - Integer Linear Programming [Lin et al., 2024]
  - Graph Coloring Problem
  - Industrial Scheduling
  - Employee scheduling [Kletzander et al., 2022, Ceschia et al., 2023a]
  - Resource-constrained project scheduling [Mischek et al., 2023]
  - Sport timetabling [Rosati et al., 2022]
  - Educational timetabling [Ceschia et al., 2023b, Bellio et al., 2021]
  - . . . and many others . . .

Local Search Elements

Key Elements

Three key elements:

1. Search Space
2. Neighborhood Relation
3. Cost Function

How these three elements are connected (e.g., which neighbor solution to select) depends upon the specific local search technique.

We use the Traveling Salesperson Problem (TSP) as an example.

Traveling Salesperson Problem (TSP)

Problem Definition:

- Given a set of cities and the distances between each pair.
- Find the shortest possible tour that visits each city exactly once and returns to the starting city.

Formally:

- Let $G = (V, E)$ be a complete graph with vertex set V (cities) and edge weights $w(u, v)$ (distances).
- Objective: Find a Hamiltonian cycle of minimum total weight.

Traveling Salesperson Problem (TSP)

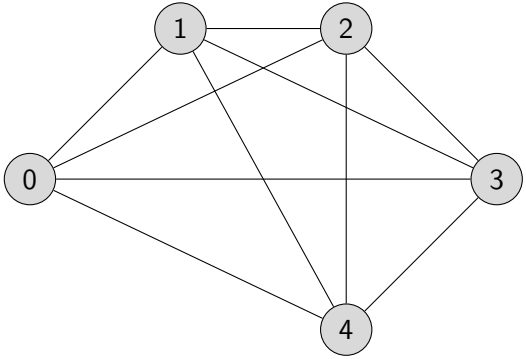


Table: Distance matrix between cities

from↓ to →	0	1	2	3	4
0	0	1	2	3	4
1	1	0	2	4	5
2	2	2	0	3	3
3	3	4	3	0	4
4	4	5	3	4	0

Traveling Salesperson Problem (TSP)

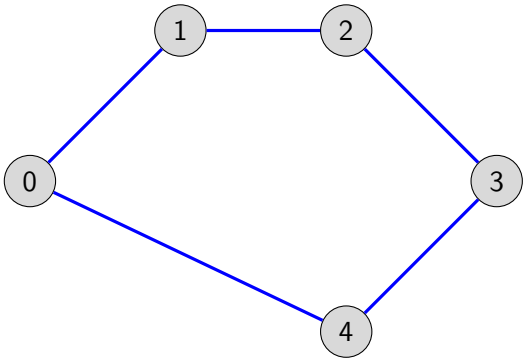


Table: Distance matrix between cities

from↓ to →	0	1	2	3	4
0	0	**1**	2	3	4
1	1	0	**2**	4	5
2	2	2	0	**3**	3
3	3	4	3	0	**4**
4	**4**	5	3	4	0

Total distance $= d_{01} + d_{12} + d_{23} + d_{34} + d_{40} = 1 + 2 + 3 + 4 + 4 = 14$

Search Space

Given a search or optimization problem P and in instance I of P, we associate to it a search space S, so that:

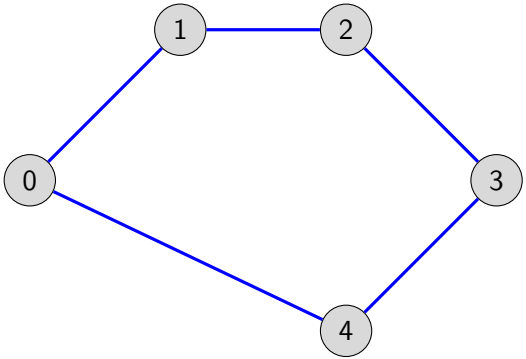
- Each element $s \in S$ is a solution of I (not necessarily feasible)
- For search problems, at least one feasible solution of I is represented in S
- For optimization problems, at least one optimal solution of I is represented in S

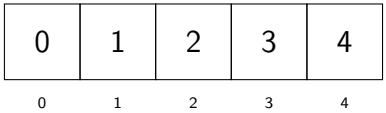
We refer to an element $s \in S$ as *state*.

⚠️ A state corresponds to a solution of I, however, not all solutions are necessarily represented by some state in the search space.
We can use the term *decision space* as a synonym.

Search Space for the TSP

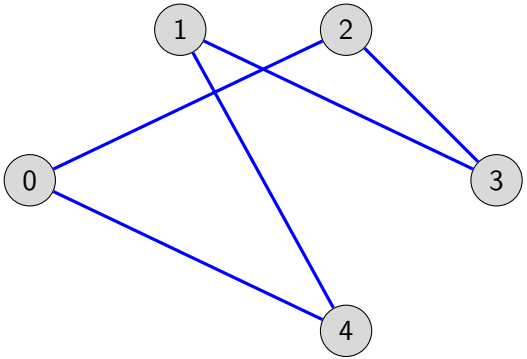
A direct representation for the TSP is an array of size $|V|$ of values representing the nodes. This array defines a permutation of the cities, specifying the order in which they are visited in the tour. The search space is composed of the set of permutations.





Search Space for the TSP

A direct representation for the TSP is an array of size $|V|$ of values representing the nodes. This array defines a permutation of the cities, specifying the order in which they are visited in the tour. The search space is composed by the set of permutations.



2	3	1	4	0
0	1	2	3	4

Search Space for the TSP

This is not the only possible representation:

- Array of successors: Position i stores the subsequent node w.r.t. node i
- Adjancency matrix: A matrix M where $M(i, j) = 1$ when we select the path from i to j
- ...

⚠ Some representations are more *intuitive* than others.

⚠ Some representations are more efficient than others.

⚠ Some representations require further specifications in the neighborhood relations/in the cost function

⚠ Some representations allow easier extensions (e.g., when adding some constraints or objective function components).

Search Space and API

- Type SOLUTION
- Data structure defining one candidate solution
- Expression of the decision space
- *Hint*: Local search works on complete solution, thus the method VALID() should consider this

Neighborhood Relation

Given an instance I of problem P and a search space S for it, we assign to each state $s \in S$ a set $\mathcal{N}(s) \subseteq S$ of neighboring states of s.

- $\mathcal{N}(s)$ is called *neighborhood*
- Each $s' \in \mathcal{N}(s)$ is called *neighbor*

The set $\mathcal{N}(s)$ does not need to be listed explicitly; instead, a set of possible *moves* can be defined.

- Moves define transitions between states
- A move m is defined by a small set of attributes that describe local modifications of s
- The state obtained by applying the move m to state s is indicated as $s \oplus m$

⚠ The search space S **must** be connected under $\mathcal{N} \rightarrow$ every state $s \in S$ can be reached from any other state $s' \in S$ through a finite sequence of moves $m \in \mathcal{N}$

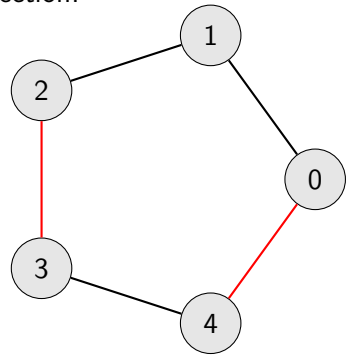
Neighborhood Relation for the TSP

- Following the TSP example, we consider a well-known neighborhood from the literature – 2-opt.

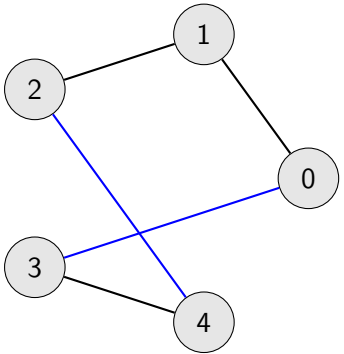
Some recent works on the topic include
[Lancia and Vidoni, 2023, da Costa et al., 2021, Eder et al., 2022]

Neighborhood Relation for the TSP

A 2-opt move removes two edges and reconnects the two paths in the opposite direction.



Before 2-opt



After 2-opt

Neighborhood Relation for the TSP

- When representing the state as a permutation, we could also try *generic* neighborhood
  - Swap: Take two items in the permutation, and swap them in the permutation.
  - Insert: Take one item, and insert it in another place of the permutation

Neighborhood Relation

- A large body of literature focuses on showing the effectiveness of some neighborhoods on various problem domains.
- However, the task of designing efficient and effective neighborhoods for novel problems is a creative task.
- Two major ways to proceed:
  - Select the solution representation ➡️ Define the neighborhoods by perturbing the representation.
  - Define your neighbors as *similar* solutions ➡️ Select the representation so to make the implementation efficient.

Neighborhood Relation and API

- Type MOVE: Identifies the changes between two neighboring solutions
  - Method APPLY_MOVE(SOLUTION): materializes the changes of a move into a given solution
  - Method OBJECTIVE_VALUE_INCREMENT (see later)
- Type NEIGHBORHOOD: Defines how to explore the neighborhood
  - Method MOVES(SOLUTION): generates all the neighbors of a solution
  - Method RANDOM_MOVES_WITHOUT_REPLACEMENT(SOLUTION): generates all the neighbors of a solution in random order
  - Method RANDOM_MOVE(SOLUTION): generates one random solution

Cost Function

The selection of the move at each step is based on the *cost function* f

- The cost function f associates at each element $s \in S$ a value $f(s)$ assessing the quality of s

⚠️ In this presentation, without loss of generality, we assume that f is non-negative and integer-valued.

Cost Function

Search Problems

- Normally, f counts the number of constraint violations, i.e., *distance to feasibility*

Optimization Problems

- Normally, f merges the distance to feasibility (if the search space also considers non-feasible solutions) and the objective function f of the problem
- The objective is usually a weighted sum, with higher weights assigned to the distance to feasibility, so as to prefer feasibility over optimality

⚠ One could also consider different ways of dealing with the cost function other than aggregation

- In some cases, one could also consider auxiliary components to help guide the search (like preferring solutions with more balanced composition).

Cost Function for the TSP

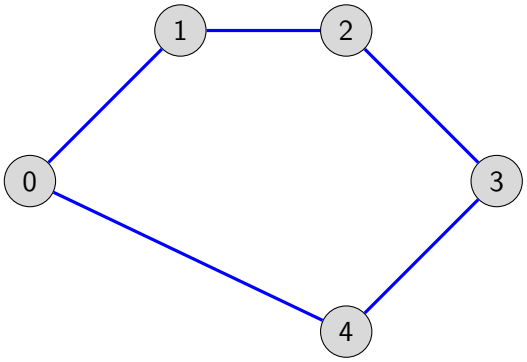


Table: Distance matrix between cities

from↓ to →	0	1	2	3	4
0	0	**1**	2	3	4
1	1	0	**2**	4	5
2	2	2	0	**3**	3
3	3	4	3	0	**4**
4	**4**	5	3	4	0

Total distance $= d_{01} + d_{12} + d_{23} + d_{34} + d_{40} = 1 + 2 + 3 + 4 + 4 = 14$

Cost Function and API

- Method OBJECTIVE_VALUE(SOLUTION)
- In the API, it is assumed that the objective function is to be minimized
- *Note*: Objective-value evaluation is required by all optimization algorithms!

Delta Cost

When defining a neighborhood, we can compute a *delta* function (also referred to as *increment*):

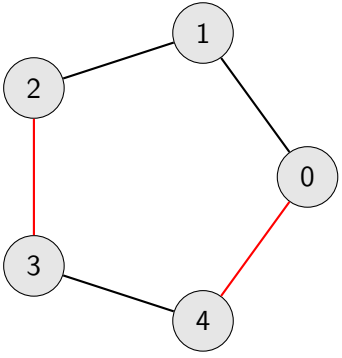
- Efficiently computes the cost difference between two neighboring solutions
- Avoids full recomputation of the objective function
- Significantly reduces evaluation time

Delta Cost for the TSP

- Consider the 2-opt move in TSP:
  - Removes two edges and reconnects the tour in a different way
- The **delta cost** can be computed using only the distances of the affected edges

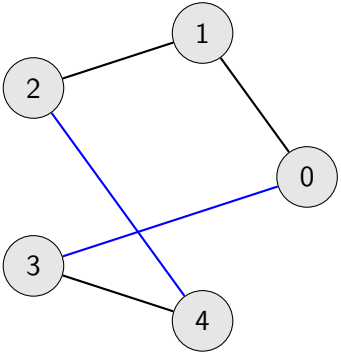
❗ No need to sum the whole tour again

Delta Cost for the TSP



Before 2-opt

$d_{01} + d_{12} + d_{23} + d_{34} + d_{40}$



After 2-opt

$d_{01} + d_{12} + d_{24} + d_{34} + d_{30}$

Delta Cost and API

- In the API, we refer to the concept of *Delta Cost* as INCREMENT.
- You need to work with OBJECTIVE_VALUE_INCREMENT(SOLUTION) within the MOVE type (class)

Initial Solution

Local Search techniques are selective techniques, i.e., they work on complete solutions

- How to generate the first solution?

Typical choices are:

- **Random Generation**
- **Greedy Constructive Heuristics**: Add a new element to the solution each time, taking the best option possible at that time
- **Any other search method**

⚠️ While already good solutions allow reaching high-quality solutions in a short time, the risk is to explore only a limited portion of the search space.

🤷 However, it depends: do you need a feasible solution? Do you have some hints on your fitness landscape? etc.

Initial Solution and API

- You can start from a random solution → you need to implement `random_solution(Problem)` method
- You can start from any other given solution → E.g., `algorithm.best_improvement(problem, solution)`

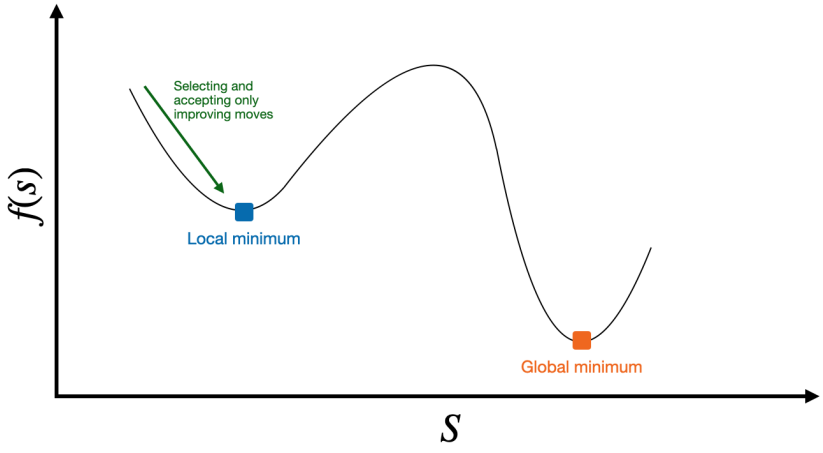
Move Selection and Move Acceptance

At each step, a single move is selected. This selection depends on the *specific* local search strategy.

⚠️ The selection of a move does not imply that the move is accepted and therefore that the state is changed. The move is subject to an *acceptance criterion*, also dependant on the local search strategy.

- Generally, improving moves are accepted
- Sometimes, also worsening moves can be accepted to escape local minima
  - State s is a local minimum if $f(s) \leq (s')\forall s' \in \mathcal{N}(s)$

Fitness and Local Minima



Stop Criterion

The stop criterion determines when the search is over.

- Natural end
  - Stagnation (e.g., best improvement)
  - Specific parameters (e.g., final temperature in Simulated Annealing)
- Given amount of time (seconds or iterations)
- Given amount of time without improvements (stagnation)
  - Search trials that are promising paths are let run longer
- Given amount of accepted moves
- Given amount of cost function evaluations (i.e., useful when calculating the cost function is expensive)
- . . . Many others . . .

Stop Criterion and API

- At this point in time, the stop criterion is algorithm-based
- First improvement and Best improvement: natural end
- Simulated Annealing: temperatures + time (in seconds)
- Iterated Local Search: time (in seconds)

Connecting Elements and Procedure

 At this point, we have all the key ingredients for understanding a Local Search algorithm.

Local Search Procedure

Algorithm 1 Abstract Local Search procedure

1: $s = \text{InitialState}(\mathcal{S})$
2: $s^* = s$
3: **while** not Stop **do**
4: $\quad m = \text{SelectMove}(s, \mathcal{N})$
5: $\quad$ **if** $\text{AcceptableMove}(s, m)$ **then**
6: $\quad\quad s = s \oplus m$
7: $\quad\quad$ **if** $F(s) < F(s^*)$ **then**
8: $\quad\quad\quad s^* = s$
9: $\quad\quad$ **end if**
10: $\quad$ **end if**
11: **end while**
12: **return** S^*

Local Search Techniques

From Paradigm to Techniques

We have defined Local Search in a general way.
Specific component implementations depend upon:

- **Problem at hand**: Initial solution, Neighborhood structure, etc.
  - You can *recycle* the same data structures for different problems (e.g., represent a solution as a permutation, swap & insert moves)
- **Specific Local Search technique**: Move selection, Move acceptance, etc.
  - The same Local Search technique can have different customization points (e.g., Simulated Annealing cooling scheme [Franzin and Stützle, 2019])

Hill Climbing

- Also known as *Iterative Improvement* or *Iterative Descent*
- Select and accept at each step a move that improves the cost function, **never** a worsening one 💡
- Variants: Steepest Descent, First Improvement Descent, Random Descent[1]

[1] Note we are alternating names such as *Climbing* and *Descending*; however, we are always refering to minimization problems.

Hill Climbing

procedure *HillClimbing*(SearchSpace $\mathcal{S}$, Neighborhood $\mathcal{N}$, CostFunction F)
    $s \leftarrow \text{InitialState}\mathcal{S}$
    while (*not Stop*)
        $m \leftarrow \mathit{SelectMove}(s, \mathcal{N})$
        $\Delta F \leftarrow F(s \oplus m) - F(s)$
        if $(\Delta F < 0)$
            $s \leftarrow s \oplus m$
    return s

Steepest Descent

- Neighborhood exploration: Exhaustive
- Stop criterion: Idle iterations (natural). It stops as soon as it reaches a local minimum.

⚠️ Issues

- Exhaustive exploration may require a lot of time
- Exhaustive exploration may raise ties (i.e., solutions in the neighborhood with equal cost). Usually, one solve ties with a random tie break.
- Trapped in local minimum (by design).

First-Improving Descent

- Neighborhood exploration: Select the first improving move. Potentially exhaustive.
- Stop criterion: Idle iterations (natural). It stops as soon as it reaches a local minimum.

⚠️ Issues:

  - To avoid bias toward some specific attributes, the exploration should start from a random move and proceed onward in a circular fashion.
  - Trapped in local minimum (by design).

Random Descent

- Neighborhood exploration: Random moves
- Acceptance criterion: Improving and sideways moves are accepted.
- Stop criterion: Local minima cannot be dected, therefore other criteria should be used.

⚠️ Issues:

  - One should design carefully how random moves are selected to ensure uniform sampling

Hill Climbing and API

- Steepest Descent
  - In the API, it is referred to as Best Improvement
  - You need to implement MOVE()
- First Improvement
  - You need to implement RANDOM_MOVES_WITHOUT_REPLACEMENT()
- Random Descent
  - You need to implement RANDOM_MOVE().

Non-Ascent Techniques

- Descent techniques: accepting only strictly improving moves
- Non-ascent techniques: accepting also moves with equal cost (*side-ways* move)
  - Allow to navigate through *plateaux*, which are areas of the search space with equal cost
  - Ability to reach states from which the cost can be decreased again.

⚠️ You might get stucked between a subset of states!!

Tabu Search

ROAR NET

⚠️ Non-ascent techniques might end up cycling between a subset of states in the plateau.

💡 Intuition: When humans solve problems they use experience and memory. Try to do the same in a local search

💡 Forbid visiting recently visited states / moves

- Use a *tabu list* to store recently visited states / moves
- At every iteration, select and accept the best (non-tabu) move in the neighborhood of the current state
- Aspiration criterion allows overriding the tabu status

📚 Originally proposed in [Glover, 1986]

🏆 For years, it has been the state-of-the-art methodology for several combinatorial optimization problems.

Tabu Search: Basic Procedure

procedure *TabuSearch*(SearchSpace $\mathcal{S}$, Neighborhood $\mathcal{N}$, CostFunction F, Parameters θ)
    $s \leftarrow InitialState(\mathcal{S})$
    $s_{best} \leftarrow s$
    $T \leftarrow InitializeTabuList(\theta)$
    while ($not\ Stop$)
        $m \leftarrow SelectMove(s, \mathcal{N})$
        $\Delta F \leftarrow F(s \oplus m) - F(s)$
        $s \leftarrow s \oplus m$
        if ($\Delta F \leq 0$)
            $s_{best} \leftarrow s$
        $UpdateTabuList(m)$
    return s_{best}

Tabu Search: Problem-independent Components

In the literature (and summarized in [Glover and Laguna, 1997]), there has been several proposals for the TS components.

- **Tabu List Management**: Fixed-length, Random bounded length, Reactive, Transition measure, etc.
- **Tabu List Item**: Move (attributes), States [Danielsen and Hvattum, 2025]
- **Aspiration Criterion**: By default, By objective, By search direction, etc.

Tabu Search: Tabu List

- Remember all states encountered so far (*strict Tabu Search*)
- Remember the last j states encountered so far
- Remember the last j moves that have been performed
- ...

Tabu Search: Tabu Tenure

- Number of iterations that a state/move/attribute is tabu
- May be static or dynamic
  - *Static*: Easy to implement; fixed number of iterations, random number of iterations, etc.
  - *Dynamic*: Vary according to the search history. E.g., based on idle iterations.

Tabu Search: Aspiration Criterion

- Condition to override the tabu status; i.e., accept a move even though it is labeled as *tabu*
- *Global best*: New best solution is found
- *Region best*: Best among the neighboring ones
- *Recent best*: Best among the recently visited ones
- ...

Simulated Annealing

ROAR NET

⚠️ Acceptance of worsening moves with a given probability to escape local minima
- The probability is set through a specific parameter, called *Temperature*
- The temperature decreases over time in the search, so as to allow balance in exploring/exploiting different parts of the search space (There exist different formulations, we will see this later)

🏆 We are going to spend some time analyzing Simulated Annealing, as it was a winning local search in many situations (e.g., ICT competition of 2021 [Rosati et al., 2022])

Simulated Annealing: History & References

⚙️ The name of the algorithm comes from annealing in metallurgy, a technique involving heating and controlled cooling of a material to alter its physical properties.

- Original Papers:
  - Kirkpatrick, Gelatt, & Vecchi (1983). *Optimization by simulated annealing*. Science, 220(4598), 671-680.
  - Černỳ (1985). *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*. Journal of optimization theory and applications, 45(1), 41-51.
- Seminal Books:
  - Van Laarhoven, & Aarts (1987). *Simulated annealing: Theory and applications*. Kluwer Academic Publisher.
  - Aarts & Korst (1989). *Simulated annealing and Boltzmann machines*. John Wiley & Sons.
- $\sim 9,700$ papers on Scopus with Simulated Annealing in the title (May 2025)

Simulated Annealing: Basic Procedure

procedure *SimulatedAnnealing*(SearchSpace $\mathcal{S}$,Neighborhood $\mathcal{N}$, CostFunction F, Parameters T_0, T_f, α, N_s)
    $T \leftarrow T_0$
    $s \leftarrow InitialState(\mathcal{S})$
    $s_{best} \leftarrow s$
    while $(T \geq T_f)$
        $n \leftarrow 0$
        while $(n < N_s)$
            $m \leftarrow RandomMove(s, \mathcal{N})$
            $\Delta F \leftarrow F(s \oplus m) - F(s)$
            if $(\Delta F \leq 0)$
                $s \leftarrow s \oplus m$
                if $(F(s) < F(s_{best}))$
                    $s_{best} \leftarrow s$
            else
                if $(RandomReal(0,1) < e^{-\Delta F/T})$
                    $s \leftarrow s \oplus m$
            $n \leftarrow n + 1$
        $T \leftarrow T \cdot \alpha$
    return s_{best}

Simulated Annealing: Problem-independent Components

[Franzin and Stützle, 2019] recently surveyed different problem-independent components for SA.

- **Initial Temperature (8 options)**: Fixed value, Proportional to initial solution cost, Maximum gap in a random walk, etc.
- **Stopping Criterion (9 options)**: Maximum time, Maximum moves, Minimum temperature, etc.
- **Temperature Restart (16 options)**: Never, Number of moves, Minimum temperature, etc.
- **Exploration Criterion (4 options)**: Random, Sequential, etc.
- **Acceptance Criterion (9 options)**: Metropolis, Geometric, etc.
- **Cooling Scheme (11 options)**: Geometric, Logarithmic, etc.
- **Temperature Scheme (9 options)**: Fixed number of moves, Number of moves proportional to the problem size, etc.

Simulated Annealing: Problem-independent Components

[Franzin and Stützle, 2019] recently surveyed different problem-independent components for SA.

- Large number of possible configurations
- Each configuration comes with parameters (to tune)
- Most important components are: Exploration and Acceptance Criterion (based on empirical evidence by [Franzin and Stützle, 2019])
- Hard to draw conclusions across different problems and datasets
- Same reflections are valid for what we observed with TS

Iterated Local Search

⚠️ Local Search algorithms may be stucked in a local minimum.

- A simple way to deal with this is to iterate the calls to the Local Search routine from a different initial configuration
- That is, you start from a sliglthy different solution w.r.t. the local optimum where the previous method did stopped

📚 Introduced by [Lourenço et al., 2010]

Iterated Local Search: Some Notes

⚠️ One should pay attention to the perturbation component:

- If too strong: the risk is to start from a random solution
- If too weak: the risk is to not be able to escape from the local minimum in which you are trapped

Other Aspects

Neighborhoods

⚠️ Sometimes navigating the search space with **one** neighborhood is not enough
- Local optima, plateaus, basins of attraction, etc.

- A viable solution is to employ *multi-neighborhoods*.
- Given K neighborhoods, where each neighborhood $k \in J$ is indicate as $\mathcal{N}_k$, the related union neighborhood $\mathcal{N}_U$ is defined as follows:

$$\mathcal{N}_U = \bigcup_{k=1}^{K} \mathcal{N}_k$$

❓ Why to use more than one neighborhood?
- A local optimum, plateau, or basin of attraction of one neighborhood is not necessarily a local optimum, plateau, or basin of attraction for another one.
- They enhance the connectivity of the search space
- Robustness of the search method to the features of the instance (and of the related landscape)

📚 Multi-neighborhoods have been largely analyzed by [Rosati, 2024]

Neighborhoods

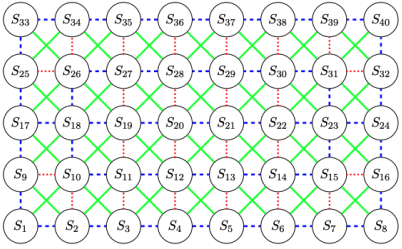


Figure: Multi-neighbohoods representation [Rosati, 2024].

- If an edge connects states s_i and s_j, it means there is a neighborhood connecting them (distinguished by color).
- Only blue neighborhood connects completely these search space.
- The green neighborhood is constrained by the choice of the initial solution
  - For instance from s_{20}, one cannot reach s_{12}, s_{19}, s_{21}, and s_{28}

Parameters and Parameter Tuning

- Both Tabu Search and Simulated Annealing are based on a set of parameters (e.g., tabu length, initial temperature, final temperature)
- Historically, parameters were manually set on the basis of a trial-and-error approach
  - Also *golden parameters* have been proposed in the years, e.g., tabu list of size 7.
- Currently, automated tools are preferred such as *irace* [López-Ibáñez et al., 2016] or *SMAC3* [Lindauer et al., 2022]
- Some studies have investigated ways of adapting parameter values during the search [Máximo et al., 2025]
- *Programming by Optimization* (PbO): Expose as many design choice as possible [Hoos, 2012]

Local Search & ROAR-NET API

- First and Best Improvements
- Simulated Annealing
- Random Descent
- Iterated Local Search
- You will recap Tabu Search with Alberto Moraglio in the next days.

Conclusion

Conclusion

- Local Search Elements: Search Space, Neighborhood, Cost Function
- Local Search Techniques: How do the elements interact
- Other aspects: Multi-neighborhood, Tuning
- What to do with your API

Acknowledgments

These slides are based on the book chapter by [Ceschia et al., 2023c] and didactic material by A.S̃chaerf. This presentation is based upon work from COST

Action Randomised Optimisation Algorithms Research Network (ROAR-NET), CA22137, supported by COST (European Cooperation in Science and Technology).

cost
EUROPEAN COOPERATION IN SCIENCE & TECHNOLOGY

Funded by the European Union

References I

Bellio, R., Ceschia, S., Di Gaspero, L., and Schaerf, A. (2021).
Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling.
Computers & Operations Research, 132:105300.

Ceschia, S., Di Gaspero, L., Mazzaracchio, V., Policante, G., and Schaerf, A. (2023a).
Solving a real-world nurse rostering problem by simulated annealing.
Operations Research for Health Care, 36:100379.

Ceschia, S., Di Gaspero, L., and Schaerf, A. (2023b).
Educational timetabling: Problems, benchmarks, and state-of-the-art results.
European Journal of Operational Research, 308(1):1–18.

Ceschia, S., Di Gaspero, L., and Schaerf, A. (2023c).
Local search.
In Minku, L. L., Cabral, G., Martins, M., and Wagner, M., editors, *Introduction to Computational Intelligence*, pages 15–28. University of Bath.

References II

da Costa, P., Rhuggenaath, J., Zhang, Y., Akcay, A., and Kaymak, U. (2021).
Learning 2-opt heuristics for routing problems via deep reinforcement learning.
SN Computer Science, 2(5):388.

Danielsen, K. and Hvattum, L. M. (2025).
Solution-based versus attribute-based tabu search for binary integer programming.
International Transactions in Operational Research, n/a(n/a).

Eder, G., Held, M., Jasonarson, S., Mayer, P., and Palfrader, P. (2022).
2-opt moves and flips for area-optimal polygonizations.
ACM J. Exp. Algorithmics, 27.

Franzin, A. and Stützle, T. (2019).
Revisiting simulated annealing: A component-based analysis.
Computers and Operations Research, 104:191–206.

References III

Glover, F. (1986).
Future paths for integer programming and links to artificial intelligence.
Computers & Operations Research, 13(5):533–549.

Glover, F. and Laguna, M. (1997).
Tabu Search.
Kluwer Academic Publishers, Boston, MA.

Hoos, H. H. (2012).
Programming by optimization.
Commun. ACM, 55(2):7080.

Kletzander, L., Mazzoli, T. M., and Musliu, N. (2022).
Metaheuristic algorithms for the bus driver scheduling problem with complex break constraints.
In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, page 232240, New York, NY, USA. Association for Computing Machinery.

References IV

Lancia, G. and Vidoni, P. (2023).

Average case subquadratic exact and heuristic procedures for the traveling salesman 2-opt neighborhood.

INFORMS Journal on Computing, 0(0):null.

Lin, P., Zou, M., Chen, Z., and Cai, S. (2024).

Parailp: A parallel local search framework for integer linear programming with cooperative evolution mechanism.

In Larson, K., editor, *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 6949–6957. International Joint Conferences on Artificial Intelligence Organization.

Main Track.

Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., and Hutter, F. (2022).

Smac3: A versatile bayesian optimization package for hyperparameter optimization.

Journal of Machine Learning Research, 23(54):1–9.

References V

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., and Stützle, T. (2016).
The irace package: Iterated racing for automatic algorithm configuration.
Operations Research Perspectives, 3:43–58.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2010).
Iterated Local Search: Framework and Applications, pages 363–397.
Springer US, Boston, MA.

Mischek, F., Musliu, N., and Schaerf, A. (2023).
Local search approaches for the test laboratory scheduling problem with variable task grouping.
Journal of Scheduling, 26(5):457–477.

Máximo, V. R., Cordeau, J.-F., and Nascimento, M. C. V. (2025).
A hybrid adaptive iterated local search heuristic for the maximal covering location problem.
International Transactions in Operational Research, 32(1):176–193.

References VI


Rosati, R. (2024).
Multi-neighborhood search for combinatorial optimization.
PhD Thesis at University of Udine.


Rosati, R. M., Petris, M., Di Gaspero, L., and Schaerf, A. (2022).
Multi-neighborhood simulated annealing for the sports timetabling competition itc2021.
Journal of Scheduling, 25(3):301–319.


Sörensen, K., Sevaux, M., and Glover, F. (2018).
A History of Metaheuristics, pages 791–808.
Springer International Publishing, Cham.