eman ta zabal zazu

Universidad del País Vasco   Euskal Herriko Unibertsitatea

Master's Degree in Computer Science

Computational Engineering and Intelligent Systems

Thesis

The "Extended Reach" Method: Exploring the neighbors of the neighbors to escape from local optima

Author

Manuel Torralbo Lezana

informatika fakultatea   facultad de informática

2022

Abstract

Local search is a widely used meta-heuristic due to its simple yet effective approach for solving computationally hard optimization problems. Despite their popularity, hill-climbing algorithms get stuck in local optima, solutions without improving neighbors form where the local search could continue. Trying to avoid this stagnant behaviour, most modern local search based algorithms have been designed to escape from local optima. Thanks to recently discovered properties of the landscapes formed by combinatorial optimization problems and neighborhood systems, a novel procedure to escape from local optima is proposed. This method, named "Extended Reach", also considers the neighbors of the neighbors of the local optima it encounters as candidate solutions to proceed to and continue the search. The "Extended Reach" method is applied to well-known permutation based combinatorial optimization problems and neighborhoods, discussing along the way the numerous developments that went into its design. The conducted experiments show promising results, successfully escaping from local optima and notably outperforming the Multi-Start heuristic, specially with instances containing a reduced number of plateaus.

Acknowledgements

Special thanks to my supervisors Jose A. Lozano and Leticia Hernando for their guidance and support, whose knowledgeable insights inspired and gave shape to this project.

Contents

Figures

Tables

Algorithms

Chapter 1

Introduction

Local search algorithms are popular and efficient methods for solving computationally hard optimization problems. At its core, local search tries to find better quality solutions in the set of neighbors of the current solution, under the established neighborhood system. In doing so, it iteratively improves, transitioning from the current solution to one of its neighbors, until a local optimum is found, that is, a solution without neighbors better than itself. Local search algorithms are considered meta-heuristic methods, as finding the best solution or global optimum is not guaranteed, and they rather return near optimal solutions: the local optimum encountered in this case.

A solution could have more than one improving solution in its neighborhood, leading to multiple paths from where the search could continue from. The most common deterministic strategies set a specific order in which to visit the neighbors and settle for the first solution with an improved quality, known as *first improvement*, or just evaluate the entire neighborhood and ensure that they transition to the best neighbor, named *best improvement*. The hill-climbing algorithms that adopt these strategies implicitly divide the search space in disjoint sets, each composed of all the solutions that lead to the same local optimum, also known as attraction basins of local optima.

The local optima are the best quality solutions found, however, since given the imposed conditions no more improvement is possible, it also supposes that local search methods inherently get stuck in these kind of solutions and their basins of attraction. As Multi-Start methods [5] suggest, one feasible approach is to restart the search all over again from a different solution every time a local optimum is encountered, trying to collect as many distinct local optima as possible from which to choose the best. Contrarily, a second variant of algorithms try to escape from the local optima, two of the most notable methods being Variable Neighborhood Search [7] and Iterated Local Search [8, 9]. The Variable Neighborhood Search switches the neighborhood system every time the search gets stagnated in a local optimum of the currently adopted neighborhood, until it meets a solution that is a local optimum in all the considered neighborhoods. On the other hand, Iterated Local Search methods incorporate a perturbation phase to rearrange the current local optimum, with the aim of generating a solution in an attraction basin leading to a different local optimum, but also maintaining as much quality as possible of the original local optimum.

It was believed that the larger the attraction basin of a local optimum, the more pronounced the perturbation had to be to escape from it. However, recent developments in the field of combinatorial landscapes [3] have shown that usually just one step in the neighborhood is enough to find a new attraction basin. Based on this knowledge, a new method named “Extended Reach” is proposed in this work. In order to escape from local optima, the “Extended Reach” method also considers the neighbors of the neighbors of the current local optimum, looking for a better solution to transition to and continue the search.

Further studies have proven, specifically for the Linear Ordering Problem and its most commonly used neighborhoods, that some of these second degree neighboring solutions will never be better than the local optimum they derive from [4]. The proposed method takes advantage of such proofs to avoid a large number of non-better solution, increasing its efficiency.

To validate the proposal, it is applied to three widely known permutation based combinatorial optimization problems: the Linear Ordering Problem, the Traveling Salesman Problem and the Quadratic Assignment Problem. The resulting implementations include multiple novel optimizations which merge the structure of the “Extended Reach” method with the specific properties of the mentioned problems. The subsequently conducted experimental study concludes that the developed procedure successfully escapes from local optima, and the comparison against the Multi-Start local search shows that the computational requirements to do so are worth the effort.

This document is structured as follows. Chapter 2 presents the three combinatorial optimization problems, while Chapter 3 defines neighborhood systems utilized to conduct the local search in these problems. Then, in Chapter 4, the proposed method is described and then applied to each particular problem, explaining the optimizations that have been discovered. Chapter 5 gives a detailed account of the computational results obtained in the experimental phase using the developed implementations. Finally, in Chapter 6, the conclusions and future work are discussed.

Chapter 2

Combinatorial Optimization Problems

The mathematical field of Combinatorial Optimization aims to solve discrete optimization problems by finding the best solution from a finite or countably infinite set of possibilities. The quality of each solution is tied to the value of the objective function f of the problem, that needs to be either maximized or minimized.

In this chapter, three particular examples of Combinatorial Optimization problems are introduced, later used to illustrate and evaluate the algorithmic developments. These are *NP-hard* problems, and thus, there are not known polynomial time exact algorithms to optimally solve all of their instances, and approximation algorithms that can find near-optimal solutions in acceptable computational time are usually used to solve them.

2.1 Linear Ordering Problem

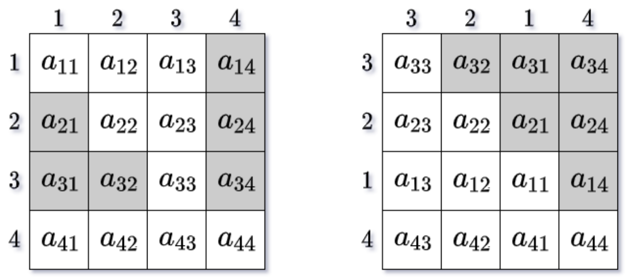
The Linear Ordering Problem (LOP) consists in finding a permutation of the set $\{1, 2, ..., n\}$ such that when both the rows and the columns of a given matrix $A = [a_{ij}]_{n\times n}$ are rearranged according to the order described by the permutation, the sum of the coefficients above the main diagonal is maximized.

Therefore, the search space of the LOP, Ω^{lop}, is the set of all possible permutations, and its size is $|\Omega^{lop}| = n!$. The objective function is:

$$f^{lop}(\sigma) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} a_{\sigma_i \sigma_j}$$

where σ_i is the i-th element of permutation σ.

The coefficient $a_{\sigma_i \sigma_j}$ of the matrix A, associated to the elements σ_i and σ_j at positions i and j of the permutation σ, contributes to the objective function only if $i < j$. That is to say, if the element σ_i is positioned before σ_j in the solution. Take for example the permutation $(3\,2\,1\,4)$ in the search space of size $n = 4$, here the coefficients that fulfill this condition are a_{32}, a_{31}, a_{34}, a_{21}, a_{24} and a_{14}, as shown in Figure 2.1.



1 2 3 4
1 a_{11} a_{12} a_{13} a_{14}
2 a_{21} a_{22} a_{23} a_{24}
3 a_{31} a_{32} a_{33} a_{34}
4 a_{41} a_{42} a_{43} a_{44}
3 2 1 4
3 a_{33} a_{32} a_{31} a_{34}
2 a_{23} a_{22} a_{21} a_{24}
1 a_{13} a_{12} a_{11} a_{14}
4 a_{43} a_{42} a_{41} a_{44}

Figure 2.1: Matrix A of the LOP for $n = 4$, in its original distribution (left) and rearranged according to solution $(3\,2\,1\,4)$ (right). The coefficients highlighted in gray add towards the cost of this permutation.

No matter the solution, an element σ_i will always be positioned before or after another distinct element σ_j. Consequently, for every pair of coefficients $a_{\sigma_i \sigma_j}$ and $a_{\sigma_j \sigma_i}$, one will stand above the main diagonal and the other below, and thus, the first one will be part of the cost and the latter will not.

2.2 Traveling Salesman Problem

Being one of the most popular and studied problems in combinatorial optimization, the Traveling Salesman Problem (TSP) consists in, given a collection of n cities and the distance between each pair of them, finding the shortest tour with the same starting and ending point that visits each city once.

Formally, the problem can be stated as follows. Let $G = (V, E, A)$ be a graph where $V = \{1, 2, ..., n\}$ is the set of vertices and $E = \{(u, v) \,|\, u, v \in V\}$ is the set of edges. Each edge (u, v) has an associated weight a_{uv} contained in the adjacency matrix $A_{n \times n}$. The classical fully connected and symmetric variant of the TSP is considered, therefore, G is a complete undirected graph and $a_{uv} = a_{vu}$.

The goal of the TSP is to find a sequence of n interconnected edges, visiting all n vertices once, and starting and ending at the same vertex, also known as a Hamiltonian cycle, that minimizes the sum of the weights tied to its edges. A solution σ of the TSP can be also represented as a permutation of V, denoting the order in which the vertices are visited, ended by going back from the vertex at the final position to that in the initial one. Considering σ_i is i-th visited vertex, the objective function to minimize is:

$$f^{tsp}(\sigma) = \left(\sum_{i=1}^{n-1} a_{\sigma_i \sigma_{i+1}} \right) + a_{\sigma_n \sigma_1}.$$

Defining the solutions in the described manner allows different permutations to outline exactly the same tour. Namely, two solutions are equivalent if either one can be transformed into the other by shifting all elements in either direction, the overflowing elements returning from the opposite end, or reversing the entire permutation, or a composition of both. Since a solution can be shifted n times until the original permutation is recovered, if its reverse is considered too, the number of equivalent solutions is $2n$, which in turn makes the size of the search space:

$$|\Omega^{tsp}| = \frac{n!}{2n} = \frac{(n-1)!}{2}.$$

2.3 Quadratic Assignment Problem

Regarded as one of the hardest problems in combinatorial optimization, the Quadratic Assignment Problem (QAP) tackles the issue of placing n facilities in n locations. The quality of the arrangement depends on the production flow between the facilities, as well as the distance separating the locations where they are established. More specifically, given an adjacency matrix $A_{n\times n}$ and a flow matrix $B_{n\times n}$, where a_{ij} is the distance from location i to location j and b_{kl} is the flow from facility k to facility l, the QAP is the problem of finding the minimum cost allocation between the n facilities and locations, taking the cost as the sum of all flow by distance products.

A solution σ of the QAP is a permutation of the set $\{1, 2, ..., n\}$, making the size of the search space $|\Omega^{qap}| = n!$, and expresses to which location each facility has been assigned, that is, facility i has been placed at location σ_i. Assuming that the main diagonals of both A and B are null, then, the objective function to minimize is:

$$f^{qap}(\sigma) = \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij}\, a_{\sigma_i \sigma_j}.$$

Chapter 3

Permutation based Neighborhood Systems

The objective function f and the search space Ω of a combinatorial optimization problem, together with a neighborhood system $\mathcal{N}$, form what is known as a landscape. The neighborhood system provides each solution $\sigma \in \Omega$ a non empty set of neighboring solutions $\mathcal{N}(\sigma)$, which is usually the set of distinct solutions that can be reached after applying once an operator $\mathcal{M}$ to σ. A solution σ^* is considered a local optimum under $\mathcal{N}$ if for every neighbor σ' in its neighborhood $\mathcal{N}(\sigma^*)$ the value of the objective function $f(\sigma^*)$ is better than or equal to $f(\sigma')$, that is to say, if σ^* has no improving neighboring solutions.

If a local optimum is strictly better than all its neighbors it is considered a strict local optimum. However, if there are neighboring solutions with the same objective function value as the local optimum, a plateau is formed, that is, a set of solutions with the same cost, all connected by the established neighborhood system modeling a network structure [3].

In this chapter, the utilized permutation based operators that define the neighborhood systems are presented first. Then, the particular properties of the neighborhoods that arise from applying these operators to the specific problems are discussed.

3.1 Operators

Adjacent Swap

The elements of a permutation σ placed at two consecutive positions i and $i+1$ are swapped. Since there are $n-1$ pairs of consecutive positions in a permutation, the size of the neighborhood of σ tied to this operator is $|\mathcal{N}_{\mathcal{S}}(\sigma)| = n-1$. Assuming that $i < n$, an *adjacent swap* operation can be expressed as:

$$\mathcal{M}_{\mathcal{S}}(\sigma; i) = (\sigma_1, ..., \sigma_{i-1}, \sigma_{i+1}, \sigma_i, \sigma_{i+2}, ..., \sigma_n).$$

2-exchange

Two elements of a permutation σ at positions i and j, not necessarily consecutive, are swapped. The size of the neighborhood tied to this operator is the number of possible combinations of two elements without repetition: $|\mathcal{N}_{\mathcal{X}}(\sigma)| = \binom{n}{2}$. With the condition that $i < j$, a 2-*exchange* operation can be written as:

$$\mathcal{M}_{\mathcal{X}}(\sigma; i, j) = (\sigma_1, ..., \sigma_{i-1}, \sigma_j, \sigma_{i+1}, ..., \sigma_{j-1}, \sigma_i, \sigma_{j+1}, ..., \sigma_n).$$

Insert

The element at position i of the permutation σ, that is, σ_i, is moved to position j. As a result, the elements placed from i to j, not including i, are shifted in the opposite direction of the insertion. Inserting an element in its same position would not alter the permutation, hence, if $i \neq j$, an *insert* operation is declared as:

$$\mathcal{M}_{\mathcal{I}}(\sigma; i, j) = \begin{cases} (\sigma_1, ..., \sigma_{i-1}, \sigma_{i+1}, ..., \sigma_j, \sigma_i, \sigma_{j+1}, ..., \sigma_n), & \text{if } i < j, \\ (\sigma_1, ..., \sigma_{j-1}, \sigma_i, \sigma_j, ..., \sigma_{i-1}, \sigma_{i+1}, ..., \sigma_n), & \text{if } i > j. \end{cases}$$

Positions i and j are not interchangeable, meaning that inserting the i-th element at j, or doing it the other way around, will not produce the same permutation in most cases. The only case where this happens is when i and j are consecutive positions. As all n elements could be inserted in the remaining $n-1$ positions, and there are $n-1$ pairs of consecutive positions, the size of the neighborhood tied to this operator is:

$$|\mathcal{N}_{\mathcal{I}}(\sigma)| = n(n-1) - (n-1) = (n-1)^2.$$

2-opt

The segment in the positions $[i, j]$ of a solution σ is reversed. Assuming that $i < j$, the size of the neighborhood tied to this operator is the number of all possible combinations of elements that define both ends of the segment: $|\mathcal{N}_{\mathcal{O}}(\sigma)| = \binom{n}{2}$. A 2-*opt* operation can be written as:

$$\mathcal{M}_{\mathcal{O}}(\sigma; i, j) = (\sigma_1, ..., \sigma_{i-1}, \sigma_j, \sigma_{j-1}, ..., \sigma_{i+1}, \sigma_i, \sigma_{j+1}, ..., \sigma_n).$$

For a permutation σ such as $(1\,2\,3\,4\,5\,6)$, the reversal of the segment going from the second position to the fifth is $\mathcal{M}_{\mathcal{O}}(\sigma; 2, 5) = (1\,5\,4\,3\,2\,6)$.

3.2 Relations between Neighborhoods

In some cases, two different operators can have the same outcome if one of them is applied, once or multiple times, in a specific manner. These equivalences suppose that given an initial solution, after applying the operators or sequences of operators, the same resulting solution is obtained.

3.2.1 *Adjacent swap* and *insert*

Any *adjacent swap* movement is an *insert* movement where the element is inserted in one of its two contiguous positions. Consequently, the *adjacent swap* neighborhood of a solution is a subset of its *insert* neighborhood, $\mathcal{N}_{\mathcal{S}}(\sigma) \subset \mathcal{N}_{\mathcal{I}}(\sigma)$, and a local optimum under the *insert* neighborhood will always be a local optimum in the *adjacent swap* neighborhood.

Moreover, as proposed in [8], an insertion can be explained as a sequence of intertwined *adjacent swaps*, that in the process also visits preceding insert movements. For example, considering the permutation $\sigma = (1\,2\,3\,4)$, if the element at the first position is to be inserted at the last one:

$$\begin{aligned}
\sigma &= (1\,2\,3\,4),\\
\mathcal{M}_{\mathcal{S}}(\sigma;1) = \mathcal{M}_{\mathcal{I}}(\sigma;1,2) &= (2\,1\,3\,4),\\
\mathcal{M}_{\mathcal{S}}\left(\mathcal{M}_{\mathcal{S}}(\sigma;1);2\right) = \mathcal{M}_{\mathcal{I}}(\sigma;1,3) &= (2\,3\,1\,4),\\
\mathcal{M}_{\mathcal{S}}\left(\mathcal{M}_{\mathcal{S}}\left(\mathcal{M}_{\mathcal{S}}(\sigma;1);2\right);3\right) = \mathcal{M}_{\mathcal{I}}(\sigma;1,4) &= (2\,3\,4\,1).
\end{aligned}$$

3.2.2 *Insert* and 2-*exchange*

Any 2-*exchange* movement can be expressed as two *insert* operations. If, as previously noted, $i < j$:

$$\begin{aligned}
\mathcal{M}_{\mathcal{X}}(\sigma;i,j) &= \mathcal{M}_{\mathcal{I}}\left(\mathcal{M}_{\mathcal{I}}(\sigma;i,j);j-1,i\right)\\
&= \mathcal{M}_{\mathcal{I}}\left(\mathcal{M}_{\mathcal{I}}(\sigma;i,j-1);j,i\right)\\
&= \mathcal{M}_{\mathcal{I}}\left(\mathcal{M}_{\mathcal{I}}(\sigma;j,i);i+1,j\right)\\
&= \mathcal{M}_{\mathcal{I}}\left(\mathcal{M}_{\mathcal{I}}(\sigma;j,i+1);i,j\right).
\end{aligned}$$

As an example of the first equivalence, the result of swapping the elements placed at positions 2 and 5 of the permutation $\sigma = (1\,2\,3\,4\,5\,6)$ is $\mathcal{M}_{\mathcal{X}}(\sigma;2,5) = (1\,5\,3\,4\,2\,6)$. The same permutation can be obtained if first the element at position 2 is inserted in position 5, $\mathcal{M}_{\mathcal{I}}(\sigma;2,5) = (1\,3\,4\,5\,2\,6)$, and then the element at position 4, originally the fifth element, is inserted in position 2, $\mathcal{M}_{\mathcal{I}}\left(\mathcal{M}_{\mathcal{I}}(\sigma;2,5);4,2\right) = (1\,5\,3\,4\,2\,6)$. The remaining equivalences follow a similar procedure.

Considering the neighborhood system where up to two *insert* operations are allowed, a local optimum under this neighborhood would also be a local optimum in both the *insert* and 2-*exchange* neighborhoods.

3.3 Relations between Neighborhoods and Problems

For any combinatorial optimization problem, since an operator maps a solution to another one, the value of the objective function will, most likely, also change. The impact on the cost can be expressed as the difference in the value of the objective function of a solution σ regarding its derived solution σ', result of an operation that turns σ into σ':

$$d(\sigma;\sigma') = f(\sigma') - f(\sigma)$$

In most cases, due to the characteristics of the operator and the problem considered, the difference in the cost can be computed more efficiently than the objective function itself. Therefore, given the value of the objective function of a solution σ, the cost of one of its neighbors σ' can be optimally retrieved as $f(\sigma') = f(\sigma) + d(\sigma;\sigma')$.

3.3.1 *Insert* Neighborhood for the LOP

An insertion will take the element σ_i and place it in a position j previous or posterior to its original position i. If $i < j$, then σ_i is inserted in a posterior position, and every element in the segment $(i, j]$ is now placed before σ_i. Accordingly, taking into account the objective function of the LOP, for every element σ_k within the segment, the coefficient $a_{\sigma_k\sigma_i}$ has to be added, and $a_{\sigma_i\sigma_k}$ subtracted, to compute the difference in the cost caused by the insertion. If $i > j$ and σ_i is inserted in a previous position instead, the entire process is done in the opposite manner. Therefore, the cost difference is:

$$d_{\mathcal{I}}^{lop}(\sigma;i,j) = \begin{cases} \sum_{k=i+1}^{j} a_{\sigma_k\sigma_i} - a_{\sigma_i\sigma_k}, & \text{if } i < j, \\ \sum_{k=j}^{i-1} a_{\sigma_i\sigma_k} - a_{\sigma_k\sigma_i}, & \text{if } i > j. \end{cases}$$

In any case, the time complexity of the above formula is $O(|i-j|) = O(n)$. Since the size of the neighborhood is of order $O(n^2)$, given the objective function value of a solution, its entire neighborhood evaluation would require $O(n^3)$ operations. However, if the neighbors are explored in the specific order described in Section 3.2.1, as proposed in [8], the time complexity can be reduced by an order of magnitude to $O(n^2)$. Namely, the *insert* neighborhood can be explored with a series of intertwined *adjacent swap* operations. An *adjacent swap* switches the position of two contiguous elements, just as an insertion where the i-th element is moved to one of its contiguous positions $j = i \pm 1$. In this particular case $|i-j| = 1$, and the difference in the cost can be computed in constant time, hence the improvement in the time complexity.

Based on this principle, the *insert* difference matrix tied to a solution σ, denoted as $D_{\mathcal{I}}^{\sigma} = [d_{ij}^{\sigma}]_{n\times n}$, is introduced. $D_{\mathcal{I}}^{\sigma}$ contains the difference in the cost of all *insert* movements subject to σ, where $d_{ij}^{\sigma} = d_{\mathcal{I}}^{lop}(\sigma;i,j)$ (see Algorithm 1).

Algorithm 1 LOP *Insert* Neighborhood Difference Matrix

```
1: function DIFFERENCE_MATRIX(SOLUTION $\sigma$, MATRIX $A$)
2:     $D^\sigma_\mathcal{I} \leftarrow$ MATRIX$[1:n, 1:n]$

3:     for ($i \leftarrow 1$; $i \leq n$; $i \leftarrow i+1$) do
4:         $d^\sigma_{ii} \leftarrow 0$

5:         for ($j \leftarrow i-1$; $j \geq 1$; $j \leftarrow j-1$) do
6:             $d^\sigma_{ij} \leftarrow d^\sigma_{i\,j+1} + a_{\sigma_i\sigma_j} - a_{\sigma_j\sigma_i}$

7:         for ($j \leftarrow i+1$; $j \leq n$; $j \leftarrow j+1$) do
8:             $d^\sigma_{ij} \leftarrow d^\sigma_{i\,j-1} + a_{\sigma_j\sigma_i} - a_{\sigma_i\sigma_j}$

9:     return $D^\sigma_\mathcal{I}$
```

Some of the significant properties of the difference matrix $D^\sigma_\mathcal{I}$ are:

- The main diagonal is null, $d^\sigma_{ii} = 0,\ \forall\ i \in \{1, 2, ..., n\}$, since inserting an element in its same position is the same as not moving it.
- All cost differences are computed using intertwined *adjacent swaps* in $O(1)$ time, requiring $O(n^2)$ for the whole matrix.
- The neighbor $\mathcal{M}^{lop}_\mathcal{I}(\sigma; i, j)$ will only be better than σ if $d^\sigma_{ij} > 0$.
- If σ^* is a local optimum, then $d^{\sigma^*}_{ij} \leq 0,\ \forall\ i, j \in \{1, 2, .., n\}$.

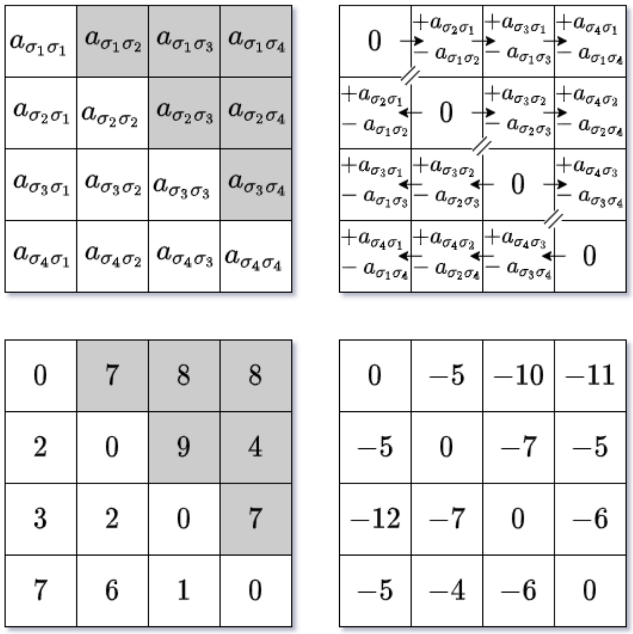
Figure 3.1 illustrates how the difference matrix is constructed in $O(n^2)$ and an example is shown.

3.3.2 2-*exchange* Neighborhood for the LOP

Considering the objective function of the LOP, and that the two exchanged elements of solution σ are σ_i and σ_j, where $i < j$, then σ_i is placed in a posterior position, and for every element σ_k in the segment (i, j) that is now before σ_i, the coefficient $a_{\sigma_k\sigma_i}$ needs to be added, and $a_{\sigma_i\sigma_k}$ subtracted. Contrarily, σ_j is moved to a previous position, and for every element σ_k now placed after it $a_{\sigma_j\sigma_k}$ has to be added and $a_{\sigma_k\sigma_j}$ subtracted. The change in position of σ_i and σ_j relative to each other only has to be accounted for once. Therefore, the difference in the cost is:

$$d^{lop}_\mathcal{X}(\sigma; i, j) = \left(\sum_{k=i+1}^{j-1} a_{\sigma_k\sigma_i} - a_{\sigma_i\sigma_k} + a_{\sigma_j\sigma_k} - a_{\sigma_k\sigma_j}\right) + a_{\sigma_j\sigma_i} - a_{\sigma_i\sigma_j}.$$

The time complexity of this function is $O(j - i) = O(n)$. The size of the 2-*exchange* neighborhood is of order $O(n^2)$, and thus, given the objective function value of a solution, the complete neighborhood evaluation requires $O(n^3)$ time.



0	7	8	8
2	0	9	4
3	2	0	7
7	6	1	0

0	−5	−10	−11
−5	0	−7	−5
−12	−7	0	−6
−5	−4	−6	0

Figure 3.1: Matrix A of the LOP for $n = 4$ according to solution σ (top left) and a visual representation of how the difference matrix D^{σ} is built (top right). An example of matrix A is shown (bottom left), as well as its corresponding difference matrix (bottom right). In this example, the provided solution is a local optimum, since no coefficient in the difference matrix is greater than 0.

Nonetheless, making use of the *insert* difference matrix described in the previous section and the observations done about the 2-*exchange* operator in Section 3.1, a better procedure of order $O(n^2)$ is proposed (see Algorithm 2). Namely, any 2-*exchange* operation can be expressed as two insert movements, and in the case of the LOP, the first insertion does not interfere in the change of the value of the objective function of the second. This supposes an equivalence in the cost difference functions between the *insert* and 2-*exchange* neighborhoods:

$$d_{\mathcal{X}}^{lop}(\sigma; i, j) = d_{\mathcal{I}}^{lop}(\sigma; i, j) + d_{\mathcal{I}}^{lop}(\sigma; j, i+1) = d_{\mathcal{I}}^{lop}(\sigma; j, i) + d_{\mathcal{I}}^{lop}(\sigma; i, j-1)$$

Following this reasoning, considering a local optimum σ^* under the *insert* neighborhood, it does not exist a 2-*exchange* movement that improves σ^*, since $d_{\mathcal{I}}^{lop}(\sigma^*; i, j) \leq 0, \; \forall\, i, j \in \{1, 2, ..., n\}$, then, no sum of two such differences could be greater than 0. So, necessarily $d_{\mathcal{I}}^{lop}(\sigma^*; i, j) + d_{\mathcal{I}}^{lop}(\sigma^*; j, i+1) \leq 0$. Therefore, for the LOP, a local optimum in the *insert* neighborhood will always be a local optimum in the 2-*exchange* neighborhood, and not necessarily the other way around.

Algorithm 2 LOP 2-*exchange* Neighborhood Evaluation

```
1: function 2EXCHANGE_BEST_IMPROVEMENT(SOLUTION $\sigma$, MATRIX $A$)
2:     $D^{\sigma}_{\mathcal{I}} \leftarrow$ DIFFERENCE_MATRIX($\sigma$, $A$)  ▷ See Algorithm 1.
3:     $best\_difference \leftarrow 0$

4:     for ($i \leftarrow 1;\ i < n;\ i \leftarrow i+1$) do
5:         for ($j \leftarrow i+1;\ j \leq n;\ j \leftarrow j+1$) do
6:             $difference \leftarrow d^{\sigma}_{ij} + d^{\sigma}_{j\,i+1}$

7:             if $difference > best\_difference$ then
8:                 $best\_difference \leftarrow difference$
9:                 $best\_i \leftarrow i$, $best\_j \leftarrow j$

10:    if $best\_difference > 0$ then
11:        $\sigma \leftarrow \mathcal{M}_{\mathcal{S}}(\sigma; best\_i, best\_j)$

12:    return $\sigma$
```

3.3.3 2-*exchange* Neighborhood for the QAP

Taking into account the objective function of the QAP, the difference in the cost of σ with respect to its neighbor $\mathcal{M}^{qap}_{\mathcal{X}}(\sigma; i, j)$, result of swapping the locations between facilities i and j, can be expressed as moving facility i to location σ_j, moving facility j to location σ_i and adjusting the flow by distance products of i and j relative to each other:

$$d^{qap}_{\mathcal{X}}(\sigma; i, j) = m(\sigma; i, j) + m(\sigma; j, i) + (b_{ij} - b_{ji})(a_{\sigma_j\sigma_i} - a_{\sigma_i\sigma_j}),$$

where
$$m(\sigma; i, j) = \sum_{\substack{k=1 \\ k\neq i,j}}^{n} b_{ik}(a_{\sigma_j\sigma_k} - a_{\sigma_i\sigma_k}) + b_{ki}(a_{\sigma_k\sigma_j} - a_{\sigma_k\sigma_i})$$

is the change in the cost of moving facility i to location σ_j. Computing the difference is, therefore, linear in time and together with the fact that the number of neighbors is of order quadratic, given the objective function value of a solution, a complete neighborhood evaluation requires $O(n^3)$.

However, as proposed in [2], if the entire neighborhood of a solution has already been examined, it is possible to explore the neighborhood of one of its neighbors in $O(n^2)$ time. Suppose that a movement exchanging facilities k and l has been done over solution σ as to obtain its neighbor $\mathcal{M}^{qap}_{\mathcal{X}}(\sigma; k, l)$. If for the sake of shortening the notation $\sigma^{kl} = \mathcal{M}^{qap}_{\mathcal{X}}(\sigma; k, l)$, then, the variation of the difference in the cost as a consequence of a second 2-*exchange* movement between facilities i and j, $\delta d^{qap}_{\mathcal{X}}(\sigma^{kl}; i, j)$, where $i, j \notin \{k, l\}$, is:

$$\begin{aligned}\delta d^{qap}_{\mathcal{X}}(\sigma^{kl}; i, j) &= d^{qap}_{\mathcal{X}}(\sigma^{kl}; i, j) - d^{qap}_{\mathcal{X}}(\sigma; i, j) \\ &= m(\sigma^{kl}; i, j) + m(\sigma^{kl}; j, i) - m(\sigma; i, j) - m(\sigma; j, i) \\ &= \delta m(\sigma^{kl}; i, j) + \delta m(\sigma^{kl}; j, i),\end{aligned}$$

where the difference in the change of the cost of moving facility i to location σ_j^{kl}, $\delta m(\sigma^{kl}; i, j)$, after simplifying the expression is:

$$\begin{aligned}\delta m(\sigma^{kl}; i, j) = m(\sigma^{kl}; i, j) - m(\sigma; i, j)&\\ = (b_{ik} - b_{il})(a_{\sigma_j^{kl}\sigma_k^{kl}} - a_{\sigma_i^{kl}\sigma_k^{kl}} - a_{\sigma_j^{kl}\sigma_l^{kl}} + a_{\sigma_i^{kl}\sigma_l^{kl}}) +&\\ (b_{ki} - b_{li})(a_{\sigma_k^{kl}\sigma_j^{kl}} - a_{\sigma_k^{kl}\sigma_i^{kl}} - a_{\sigma_l^{kl}\sigma_j^{kl}} + a_{\sigma_l^{kl}\sigma_i^{kl}}).&\end{aligned}$$

Locations i and j must be distinct to k or l so that the assigned facilities remain the same in both σ and σ^{kl}. Otherwise, the simplification is not possible and the cost difference has to be computed with the original formula in linear time. Therefore, provided the difference matrix $D_{\mathcal{X}}^{\sigma} = [d_{ij}^{\sigma}]_{n\times n}$ of solution σ, where $d_{ij}^{\sigma} = d_{\mathcal{X}}^{qap}(\sigma; i, j)$, the difference matrix of σ^{kl} itself is obtained as follows:

$$d_{ij}^{\sigma^{kl}} = \begin{cases} d_{ij}^{\sigma} + \delta d_{\mathcal{X}}^{qap}(\sigma^{kl}; i, j), & \text{if } i, j \notin \{k, l\},\\ d_{\mathcal{X}}^{qap}(\sigma^{kl}; i, j), & \text{otherwise.}\end{cases}$$

computing the k and l columns and rows of the matrix using the standard difference function, requiring $2n-3$ function calls in total. The sum rule dictates that these calls do not affect the time complexity when it comes to computing the whole matrix, which remains of order $O(n^2)$.

3.3.4 2-*opt* Neighborhood for the TSP

In the 2-*opt* neighborhood for the TSP, two edges of a solution σ, (u_1, v_1) and (u_2, v_2), each formed by two consecutive vertices in the positions $i-1$, i, j and $j+1$, respectively, are replaced by the edges (u_1, u_2) and (v_1, v_2).

As shown in Figure 3.2, the difference in the cost of the neighbor $\mathcal{M}_{\mathcal{O}}^{tsp}(\sigma; i, j)$ with respect to the original solution σ can be obtained in constant time, just adding the weights of the new edges and subtracting those of the old ones:

$$d_{\mathcal{O}}^{tsp}(\sigma; i, j) = a_{\sigma_{i-1}\sigma_j} + a_{\sigma_i\sigma_{j+1}} - a_{\sigma_{i-1}\sigma_i} - a_{\sigma_j\sigma_{j+1}},$$

hence, given the objective function value of a solution, its complete neighborhood evaluation is of order $O(n^2)$.

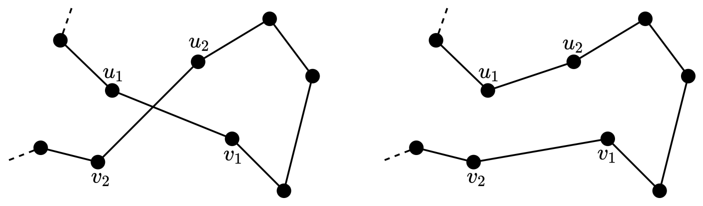


Figure 3.2: Layout of a tour of the TSP before (left) and after (right) applying a 2-*opt* movement. Edges (u_1, v_1) and (u_2, v_2) are removed and substituted for (u_1, u_2) and (v_1, v_2).

Chapter 4

The "Extended Reach" Method

A local search algorithm, in essence, starts from an initial solution σ and searches in its neighborhood $\mathcal{N}(\sigma)$ for a solution with better cost. If such neighbor σ' is found, the current solution σ is replaced and the search continues by repeating this process. Otherwise, the current solution is a local optimum σ^* by definition, since it does not have a better neighboring solution.

The deterministic strategies that dictate the path the algorithm will take are usually either *first improvement*, in which by exploring the neighborhood in a fixed order, the current solution is replaced by the first improved neighbor found, or *best improvement*, in which the current solution is replaced by its best neighbor, only if it is better. In the event of a tie, where multiple neighbors have the best value of the objective function, the solution visited first is kept. Starting from a given solution σ, the algorithms adopting these strategies will always end in the same local optimum σ^*, passing trough the same series of intermediate solutions. The attraction basin of a local optimum σ^*, is composed of all the solutions that lead to the same local optimum σ^*, after applying one of such hill-climbing algorithms.

Due to an erroneous intuition, the past understanding was that the larger the size of an attraction basin, the further away a solution had to be from the local optimum in order to escape from it, measuring the distance between two solutions as the number of operations required to get from one the other. For this reason, some Iterated Local Search algorithms escape from local optima applying numerous times the same operator, which could be the same operator as the one used in the local search or a different one.

However, as discovered in [3], checking the neighbors of a local optimum is usually enough to find a solution belonging to the attraction basin of a different local optimum. Moreover, if the *best improvement* strategy is used, the new local optimum will always be better than the current one.

Inspired in these ideas, a novel approach to escape from local optima is proposed: the "Extended Reach" method (ER). Once a local optimum is found by means of a regular local search, ER also evaluates the neighborhoods of the neighboring solutions of the local optimum. If a neighbor of a neighbor better than the local optimum is found, the method moves to the improved solution, an event known as a jump, and continues the search.

4.1 General Scheme

The initial structure follows the same path as an standard local search would, moving from neighbor to neighbor until it arrives to a local optimum, then, instead of giving up the search, the method also explores the neighbors of the neighbors of this local optimum, also known as second degree neighbors.

If a neighbor of a neighbor better than the current local optimum is found, similar to the *first improvements* and *best improvement* strategies, two different approaches are possible. The first is to jump directly to this new improving solution. The second is to complete the neighborhood evaluation of the current neighbor and take the best possible jump, if more than one are possible. In the context of this project, the latter approach is used.

The process starts over from the new solution, proceeds to the basic local search and explores again the neighborhoods of the neighbors of the resulting local optimum. This is repeated until none of the evaluated second degree neighbors overcomes the current local optimum and no jump is done.

4.2 Sorting the Neighbors

The total number of second degree neighbors for a single solution is the size of the neighborhood in question squared. For example, considering the 2-*exchange* neighborhood with a size of order $O(n^2)$, there are approximately n^4 ways to rearrange a solution if two operations are allowed. In the worst case scenario, when a local optimum has no improving second degree neighbors, evaluating all these possible solutions would be too computationally costly. Therefore, the ER method requires improvements to narrow the search and increase the odds of performing a jump. In that regard, a point of interest in the design of the ER method is to test the impact in the performance caused by the order in which the neighbors are visited at the time of exploring their neighborhoods.

To that end, a preliminary experiment is conducted strictly for the LOP. Three different versions of the ER method are considered, each visiting the neighbors in a different order regarding their objective function value: from best to worst, from worst to best, and randomly. These ER versions are implemented in a Multi-Start structure using the *best improvement* strategy, repeatedly starting from random solutions once a local optimum with no better second degree neighbor is found. In order to test the implementations, 10 instances of size $n = 50$ are created for the LOP, the matrix entries being sampled from a uniform distribution in the interval $[0, 10000]$ and each instance running 10 different times with a limit of 5000^2 objective function calls per execution.

The performance metrics used are the relative percentage error with respect to best known solutions (the best solutions obtained in the experiment in this case), the number of local optima encountered and the number of jumps performed at each execution, that is, the number of times a second degree neighbor better than the local optimum is found. The results are shown in Figure 4.1.

(per execution)	Order of neighbors		
	Best to worst	Worst to best	At random
Relative % error	**0.45**	*0.98*	0.58
	0.37	0.53	0.41
Local optima encountered	**10.64**	*2.01*	6.17
	4.03	0.1	1.9
Jumps performed	**8.64**	*0.97*	4.73
	4.03	0.22	1.93

Table 4.1: Performance comparison for the LOP of visiting the neighbors in the ER method from best to worst, worst to best and in random order. The upper number in every cell denotes the mean the and the bottom number the standard deviation. The best results are shown in bold letters and worst results in italic.

The experiment concludes that sorting the neighbors of the local optima regarding their quality, from best to worst, and visiting them in that order gives the best results. Even if the worst neighbors have more room for improvement, in absolute terms, the neighbors of the best neighbors are more likely to be of better quality. Doing the search in this order allows ER to increase the number of jumps it performs, halting the evaluation of the second degree neighbors and increasing its efficiency.

This initial experimental study is solely done for the LOP, however, there is no reason to believe that the ER method would behave differently in the TSP and QAP. The first operation applied to a local optimum is known to worsen, or in the best case maintain, the quality of the resulting neighbor. For a second degree neighbor to be better than the local optimum, the second operation needs to positively offset and exceed the difference in the cost produced by the first one. Intuitively, the worse a neighbor of a local optimum is, the more unlikely it will be for one of its second degree neighbors to compensate the negative impact and improve over the local optimum.

4.3 Stopping Criterion

Even if the neighbors are visited from best to worst, if a local optimum with no improving second degree neighbors is reached, the ER method would still be required to evaluate all the neighbors of the neighbors. The simplest approach is to just consider the neighborhoods of the k best neighbors of each local optimum.

In order to tune parameter k for each combinatorial optimization problem, an additional preliminary experiment is conducted. In this case, information regarding the jumps is sampled using the same algorithm as the one described in the previous section, visiting the neighbors from best to worst. Each time a jump occurs, the rank of the neighbor currently being visited is stored, that is, the position of the neighbor once sorted from where the jump takes place. In this sampling process, the instances used are of size $n = 100$ and composed of entries obtained from uniform distributions, in the interval $[0, 10000]$ for the LOP and TSP, and in the interval $[1, 100]$ for both matrices of the QAP. Instances are randomly generated and run single time with a limit of $1000n^2$ objective function calls until 100000 jumps in total are sampled for each problem.

With this amount of samples it is possible to estimate where most jumps occur and asses when to stop evaluating the second degree neighbors if an improving one has not been found yet. To that end, the cumulative density of the jumps with respect to the rank of the neighbors is shown in Figure 4.1. The plots reveal that the jumps are really condensed in the QAP instances, most of them taking place in the first 25 best neighbors, whereas in the TSP instances, the jumps are a lot more disperse and are still likely to happen even in the 100-th ranked neighbor. The LOP instances are somewhere in between, most of the jumps happening in the first 50 neighbors.

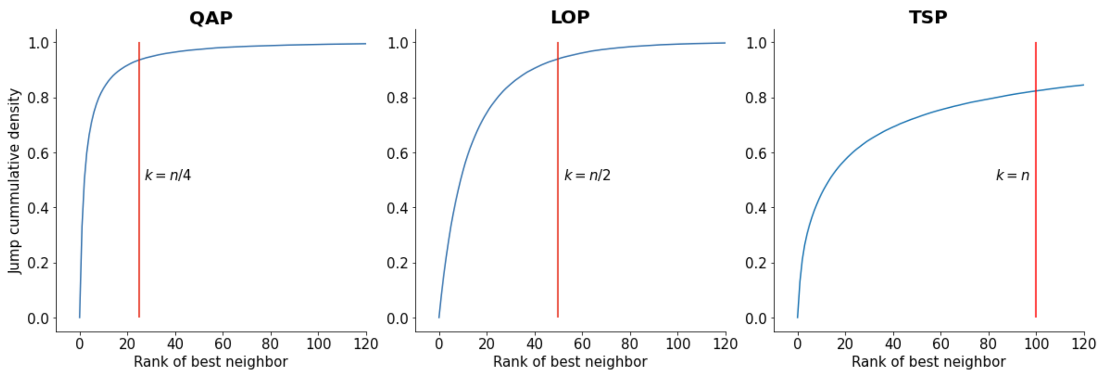


Figure 4.1: Cumulative density of the jumps with respect to the rank of the neighbors if visited from best to worst. The vertical red line shows the position in which it has been decided ER will stop evaluating the neighborhoods of the neighbors of the local optima.

It has been decided to keep parameter k proportional to the size of the instance, of order $O(n)$, trying to capture 90% to 95% of all jumps. According to results observed in Figure 4.1, the selected values of parameter k for each of the combinatorial optimization problems are: $k = n/2$ for the LOP, $k = n$ for the TSP, and $k = n/4$ for the QAP.

Note that introducing parameter k to the method supposes that ER will not only stop once a local optimum with all its neighbors of neighbors worse than itself is found, since a large number of solutions will be overlooked.

4.4 Template Pseudocode

Once the sorting criterion of the neighbors and parameter k are introduced to ER method, the resulting pseudocode is presented in Algorithm 3.

Algorithm 3 The "Extended Reach" Method

```
1:  procedure EXTENDED_REACH(SOLUTION $\sigma$, INTEGER $k$)
2:      repeat
3:          $jump \leftarrow$ FALSE
4:          $\sigma \leftarrow$ LOCAL_SEARCH($\sigma$)
5:          $neighbors \leftarrow$ K_BEST_NEIGHBORS($\sigma$, $k$)

6:          for all $\sigma' \in neighbors$ do
7:              for all $\sigma'' \in \mathcal{N}(\sigma')$ do
8:                  if $\sigma''$ is better than $\sigma$ then
9:                      $\sigma \leftarrow \sigma''$
10:                     $jump \leftarrow$ TRUE
11:             if $jump$ = TRUE then break
12:
13:     until $jump$ = FALSE

14:     return $\sigma$
```

At line 4, the function LOCAL_SEARCH(), as the name suggests, performs a regular local search starting from the solution given as a parameter and returns the resulting local optimum. At line 5, the function call computes a list with the k best neighbors of the local optimum and sorts them regarding their quality, from best to worst.

The time complexity of a single ER iteration is $O\left(k \times |\mathcal{N}(\sigma)| \times O(f)\right)$, where $|\mathcal{N}(\sigma)|$ is the size of the neighborhood and $O(f)$ is the computational cost of evaluating a solution. If a more efficient cost difference function is available it can be used instead.

4.5 Problem-specific Optimizations

Particular properties arise from combining information regarding the structure of the problems and their landscapes with those of the ER method. Based on mathematical proofs, the proposed optimizations safely discard second degree neighbors that will never be better than the local optimum without the need of evaluating them. Such improvements may also use information regarding the local optima or already conducted evaluations to further speed up the search.

4.5.1 Optimizations for the LOP

The ER implementation for the LOP utilizes the *insert* neighborhood to conduct the local search. The insights elaborately described in [4] allow to identify solutions necessarily worse than a known local optimum without the need of computing their value of the objective function. Only the neighbors of the neighbors of a local optimum whose two consecutive insertions cross each other, meaning that the first *insert* movement disturbs the difference in the cost of the second, could possibly produce a solution that is better.

Suppose that the algorithm has arrived to a local optimum σ^* and the neighborhood of one of its neighbors $\mathcal{M}_{\mathcal{I}}^{lop}(\sigma^*; k, l)$, obtained after inserting the element at position k in l, is being explored. Iterating over the positions i and j that give way to a second *insert* movement that crosses the first one, and using the difference matrix of σ^*, presented in Section 3.3.1, it is possible to obtain the difference in the cost caused by both insertions in constant time.

Take for example the solution illustrated in Figure 4.2, where the first insertion has already been carried out, moving the element at k to a posterior position l, shifting in the process the elements in the positions $(k, l]$ one place towards the start. It is known that the difference in the cost produced by such movement is $d_{\mathcal{I}}(\sigma^*; k, l)$.

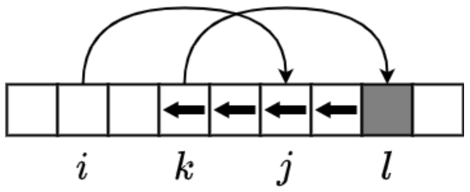


Figure 4.2: Representation of two consecutive crossing *insert* movements where the first insertion has already concluded.

Additionally, a second *insert* movement is going to take place moving the element at i, anterior to k, to a position j between k and l. As mentioned, $\mathcal{M}_{\mathcal{I}}^{lop}(\sigma^*; k, l)$ has been disrupted by the first movement and the element that is now at j was beforehand at $j+1$, which indicates that the difference in the cost of completing the second insertion is close to $d_{\mathcal{I}}(\sigma^*; i, j+1)$. However, this difference is taking into account the *adjacent swap* between the elements placed at i and k, when the element that was at k is now actually in l, a position posterior to the insertion point j. If this *adjacent swap* is corrected the total cost difference of consecutively applying both insertions is:

$$d_{\mathcal{I}}(\sigma; k, l) + d_{\mathcal{I}}(\sigma; i, j+1) + a_{\sigma_i \sigma_k} - a_{\sigma_k \sigma_i}.$$

Just like this example there are other seven cases of possible two crossing *insert* movements, for a total of eight, each with its particular configuration of the movements and adjustment to the difference in the cost (see Table 4.2).

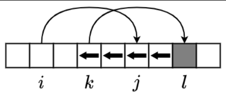
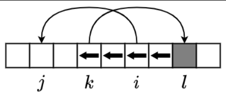
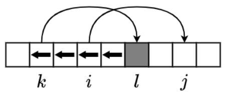
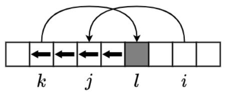
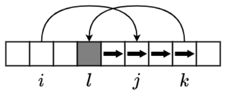
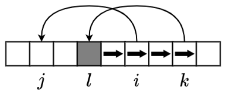
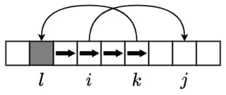
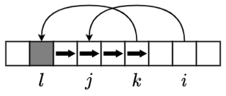
Representation	Order	Cost Difference
	$i < k \le j < l$	$d_{\mathcal{I}}(\sigma; k, l) + d_{\mathcal{I}}(\sigma; i, j+1) + a_{\sigma_i \sigma_k} - a_{\sigma_k \sigma_i}$
	$j < k \le i < l$	$d_{\mathcal{I}}(\sigma; k, l) + d_{\mathcal{I}}(\sigma; i+1, j) + a_{\sigma_k \sigma_{i+1}} - a_{\sigma_{i+1} \sigma_k}$
	$k \le i < l \le j$	$d_{\mathcal{I}}(\sigma; k, l) + d_{\mathcal{I}}(\sigma; i+1, j) + a_{\sigma_k \sigma_{i+1}} - a_{\sigma_{i+1} \sigma_k}$
	$k < j \le l < i$	$d_{\mathcal{I}}(\sigma; k, l) + d_{\mathcal{I}}(\sigma; i, j+1) + a_{\sigma_i \sigma_k} - a_{\sigma_k \sigma_i}$
	$i < l \le j < k$	$d_{\mathcal{I}}(\sigma; k, l) + d_{\mathcal{I}}(\sigma; i, j-1) + a_{\sigma_k \sigma_i} - a_{\sigma_i \sigma_k}$
	$j \le l < i \le k$	$d_{\mathcal{I}}(\sigma; k, l) + d_{\mathcal{I}}(\sigma; i-1, j) + a_{\sigma_{i-1} \sigma_k} - a_{\sigma_k \sigma_{i-1}}$
	$l < i \le k < j$	$d_{\mathcal{I}}(\sigma; k, l) + d_{\mathcal{I}}(\sigma; i-1, j) + a_{\sigma_{i-1} \sigma_k} - a_{\sigma_k \sigma_{i-1}}$
	$l < j \le k < i$	$d_{\mathcal{I}}(\sigma; k, l) + d_{\mathcal{I}}(\sigma; i, j-1) + a_{\sigma_k \sigma_i} - a_{\sigma_i \sigma_k}$

Table 4.2: Visual representation regarding the order of the positions and consequent difference in the cost of the eight different cases of two crossing *insert* movements.

Using the presented formulas, the *insert* difference matrix $D_{\mathcal{I}}^{\sigma}$ of solution σ (see Section 3.3.1) is enough to evaluate any of the second degree neighbors of σ in constant time. Regarding the computational time saved by avoiding the non-better solutions, in the worst case scenario, when the number of elements between the positions k and l of the first insertion is $n/2$, the number of second degree neighbor to evaluate is of order $O(n^2/2)$, still of the same magnitude as the dimension of the *insert* neighborhood, but approximately sparing half of the evaluations, nonetheless.

4.5.2 Optimizations for the TSP

The 2-*opt* is used to conduct the local search in the ER implementation for the TSP. Due to the characteristics of the problem and the neighborhood itself, it is also possible to rule out some neighbors of the neighbors known to be worse than the current local optimum. The proposed strategy is inspired by the insights for the LOP described in the previous section and to the best of our knowledge, the presented properties have never been analyzed before, however, they closely resemble the 3-*opt* and 4-*opt* transformations widely discussed in the literature.

Suppose that σ^* is a local optimum under the 2-*opt* neighborhood and that σ' is one of its neighbors, which has been obtained by replacing the edges (w_1, z_1) and (w_2, z_2), with (w_1, w_2) and (z_1, z_2). Considering the cyclic nature of a tour, two separate segments can be distinguished in σ' regarding the former orientation that the edges contained in them had in σ^*. In the segment going from vertex w_2 to z_1, the edges have been reversed and are now traversed in the opposite direction. In the other segment, which goes from z_2 to w_1, the edges have kept the same orientation as in the original solution.

If a second 2-*opt* movement would be done over σ' with the edges (u_1, v_1) and (u_2, v_2), in the event that both edges belonged to the same segment, meaning that the two were reversed or remained identical after the first movement, the new edges (u_1, u_2) and (v_1, v_2) would be introduced, resulting in the same exchange as if it had been done over σ^*, already known for not improving the cost. Contrarily, provided that between (u_1, v_1) and (u_2, v_2) one had kept its orientation and the other had been reversed, the edges to be added would be (u_1, v_2) and (v_1, u_2), since the previously described exchange would no longer return a valid tour (see Figure 4.3). The impact in the cost would also differ and needs to be explored in order to know the outcome.

Therefore, the reversed edges of a neighbor σ', product of a 2-*opt* movement applied to a local optimum σ^*, have to be paired with those that have kept their orientation, or vice versa, as to produce by a second movement a neighbor of a neighbor possibly better than σ^*. In the worst possible scenario, when the number of edges between the two exchanged edges in the first movement, not including (w_1, w_2) and (z_1, z_2), is $(n-2)/2$, the number of possible combinations is $((n-2)/2)^2$. On the other hand, each of the two edges added by the first movement, being new, could be paired with any other edge with the exception

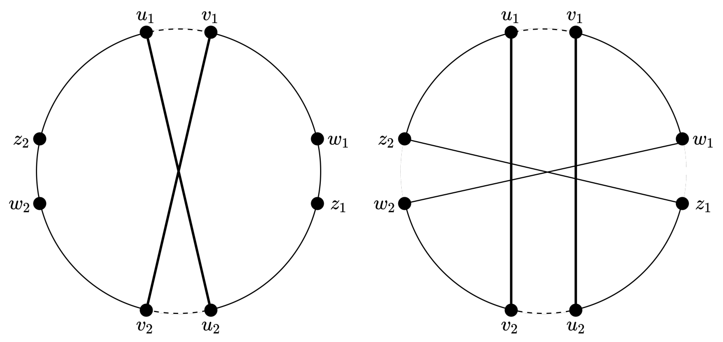


Figure 4.3: Ordinary 2-*opt* movement between edges (u_1, v_1) and (u_2, v_2) (left), and the same movement if previously another had reversed one of the edges (right). The former adds (u_1, u_2) and (v_1, v_2), the latter (u_1, v_2) and (v_1, u_2).

of themselves, their two adjacent and, in addition, the other new edge, since doing so would revert back to σ^* otherwise. The number of neighbors to be explored is decreased to:

$$\left(\frac{n-2}{2}\right)^2 + 2(n-4) = \frac{n^2}{4} + n - 7,$$

still of order $O(n^2)$, but nearly reducing the number of evaluations required by half as the size n tends to infinity.

4.5.3 Optimizations for the QAP

The 2-*exchange* neighborhood is used in the ER implementation for the QAP. One of the points of interest regarding this problem is that, unlike the other implementations discussed, a mathematical basis has not been discovered to prove that a neighbor of a neighbor has worse or equal cost to the local optimum, not leaving another option but to explicitly evaluate the whole neighborhood of each neighbor.

Nevertheless, an improvement in the search is still possible using the concepts proposed in [2], allowing to evaluate the majority of the neighbors of the neighbors in constant time. Suppose that the ER method during the the local search has arrived to a local optimum σ^*. Regardless of the strategy used to evaluate the neighborhood, the objective function value of all the neighboring solutions have already been computed to prove that σ^* is indeed a local optimum. These evaluations can be used to fill the 2-*exchange* difference matrix $D_{\mathcal{X}}^{\sigma^*}$ of σ^*, described in Section 3.3.3, and this matrix in turn can be used to evaluate any of the second degree neighbors of σ^* in constant time, required by the ER method.

Adopting these changes, the complete neighborhood evaluation of the neighbors of a local optimum is reduced to $O(n^2)$, again, no matter the strategy used to scan and exit a neighborhood.

Chapter 5

Experimental Study

The goal of this chapter is to evaluate the proposed method using different randomly generated instances and benchmark libraries, comparing it with the widely known Multi-Start (MS) method to test whether it escapes from local optima, and if it does, if the performance increase outweighs the computational requirements of doing so.

All the results shown in this chapter were achieved running C++ programming language implementations of the developed algorithms in a custom desktop computer with a Ryzen 7 2700X at 4.3GHz overclocked with Precision Boost Overdrive and 16GB of RAM under Linux Debian 11.

5.1 Instances and Benchmark Libraries

150 instances were created for each combinatorial optimization problem, 50 instances each of sizes $n = 100$, 150 and 250:

- **LOPGEN**: Randomly generated instances for the LOP composed of a single matrix where each coefficient is an integer extracted from an uniform distributions between 0 and 1000.
- **TSPGEN**: Randomly generated fully connected and symmetric graphs for the TSP whose edges have uniformly distributed integer weights between 0 and 1000.
- **QAPGEN**: Randomly generated instances for the QAP in which the distance and flow matrix entries are integers randomly chosen from the interval [1,100].

These randomly generated instances have a wide range of values to reduce the odds of tied solutions and avoid finding a plateaus.

On the other hand, to evaluate the performance in more challenging and realistic instances, the following benchmark libraries are used:

- **xLOLIB** [8]: Matrices derived from the smaller instances of the LOLIB library. 39 instances each for size $n = 150$ and 250 for a total of 78.
- **TSPLIB** [6]: Instances for the symmetric TSP from various sources and types. Only the graphs with less than 1000 vertices are used, 78 instances in total.
- **QAPLIB** [1]: Problem instances generated by several researchers for their own testing purposes, 136 instances with sizes up to 256.

5.2 Evaluation criterion

For the sake of a fair comparison, all algorithms run for a limited number of evaluations per execution. Every time the objective value of a solution is computed, the number of performed evaluations is increased until the maximum allowed is reached and the stopping criterion is met.

Not all functions require the same computational time to evaluate a solution, and thus, the number of evaluations required for each formula is proportional to its asymptotic complexity (see Table 5.1).

	Formula	Evaluations
Implementations for the LOP	f^{lop}	$n(n-1)/2$
	$d_{\mathcal{I}}^{lop}$	1
Implementations for the TSP	f^{tsp}	n
	$d_{\mathcal{O}}^{tsp}$	1
Implementations for the QAP	f^{qap}	n^2
	$d_{\mathcal{X}}^{qap}$	n
	$\delta d_{\mathcal{X}}^{qap}$	1

Table 5.1: Number of evaluations required for each of the formulas used by the developed implementations.

5.3 Behaviour of the "Extended Reach" Method

The key point of this proposal is to evaluate whether the ER method effectively escapes from local optima. In order to study its behaviour, a Multi-Start algorithm including the ER method is used, named MS-ER (see Algorithm 4). MS-ER restarts the search from a random solution every time the ER method is unable to escape from a local optimum, until the maximum number of evaluations is reached, as discussed in the previous section. The local search uses the *best improvement* strategy to evaluate the neighborhood.

Algorithm 4 The MS-ER algorithm

```
1: function MS-ER(INTEGER $k$)
2:     $\sigma \leftarrow$ RANDOM_SOLUTION()

3:     while $n\_evals < max\_evals$ do
4:         $\hat{\sigma} \leftarrow$ RANDOM_SOLUTION()
5:         $\hat{\sigma} \leftarrow$ EXTENDED_REACH($\hat{\sigma}, k$)    ▷ See Algorithm 3.
6:         if $\hat{\sigma}$ is better than $\sigma$ then $\sigma \leftarrow \hat{\sigma}$

7:     return $\sigma$
```

With this in mind, MS-ER is implemented for the three combinatorial optimization problems. All implementations, when possible, use the most efficient formulas to evaluate the explored solutions. In addition, the MS-ER algorithms also include the optimizations that arise from combining the particular properties of each problem with the ER method, discussed in Section 4.5, and parameter k is set as described in Section 4.3. The implementations ran with a limit of $1000n^2$ evaluations and 20 executions were conducted in all instances.

The number of jumps per start is one of the most relevant metrics regarding the efficacy of the ER method, since it counts how many local optimum have been successfully avoided since the algorithm started from a random solution until it found a local optimum it could not escape. A summary of the average jumps per start is shown in Figure 5.1.

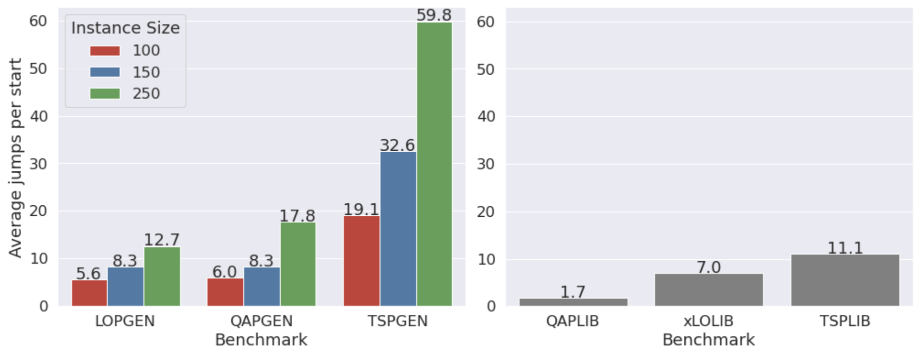


Figure 5.1: Average jumps per start of the MS-ER algorithm in the randomly generated instances regarding their size (left), and in the benchmark libraries (right).

The results reveal than the ER method escapes from local optima in all problems and instance types. It is worth to mention, that the number of jumps increase with the size of the instances, concluding than the larger the instance the more likely it is to find a second degree neighbor better than the local optimum.

According to the results observed in Figure 5.1, the number of escaped local optima in the randomly generated instances and the benchmark libraries, for the same optimization problem, are significantly different. For example, in the QAPGEN instances of $n = 100$, 6 local optima are avoided on average per start, however, in the QAPLIB, the number is reduced to 1.7. The decrease in the number of jumps is also apparent in all benchmark libraries, making them harder instances for the ER method.

The number of consecutive local optima are also counted, that is, the number of times that after escaping from a local optimum the solution to which the algorithm has jumped to is also a local optimum. The proportion between the consecutive local optima and the total number of jumps could give a sense regarding the closeness between the local optima.

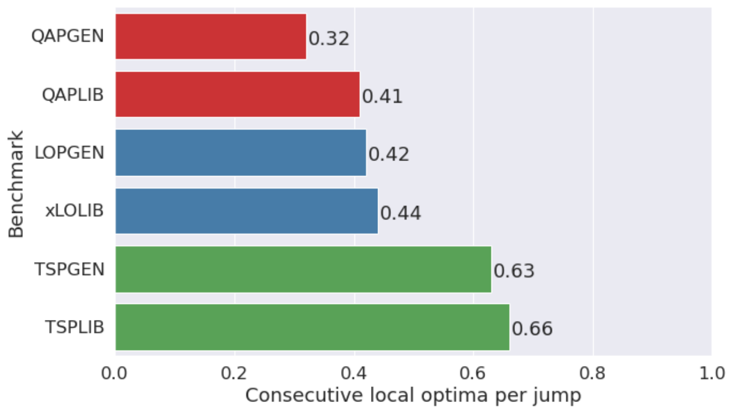


Figure 5.2: Consecutive local optima per jump regarding each instance type.

Figure 5.2 shows that the explored local optima are extremely close between each other, in the best case scenario, nearly one third of the jumps that occur in the QAPGEN instances go directly to another local optimum, two thirds in the case of the TSPLIB benchmark library. This shows that once the ER method as performed the first jump the next local optima are found rapidly and in a short amount of steps.

5.4 Comparing to the Multi-Start Heuristic

Another critical part of the project is to check if the ER method escapes from local optima in acceptable computational times. In order to do so MS-ER, already described in the previous section and Algorithm 4, is compared to a regular Multi-Start heuristic (MS) using the *best improvement* strategy. For the three combinatorial optimization problems both algorithms are tested. All algorithms ran with a limit of $1000n^2$ evaluations and 20 executions were conducted per instance.

The performance is measured in terms of the relative percentage error with respect to the best known solution, the best solutions obtained in this work, in the case of the randomly generated instances, and the best results listed in the literature, in the case of the benchmarks libraries. For each combinatorial optimization problem, LOP, TSP and QAP, the obtained results are shown as box-plots according to the size of the instances in Figures 5.3, 5.4 and 5.5, respectively.

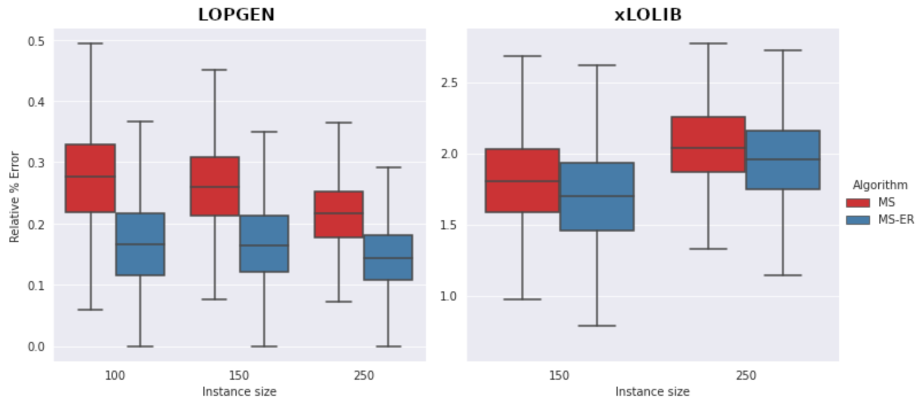


Figure 5.3: Performance comparison between Multi-Start (MS) and Multi-Start plus Extended Reach (ER-MS) implementations for the LOP. The relative percentage error is obtained with respect to the best known solutions.

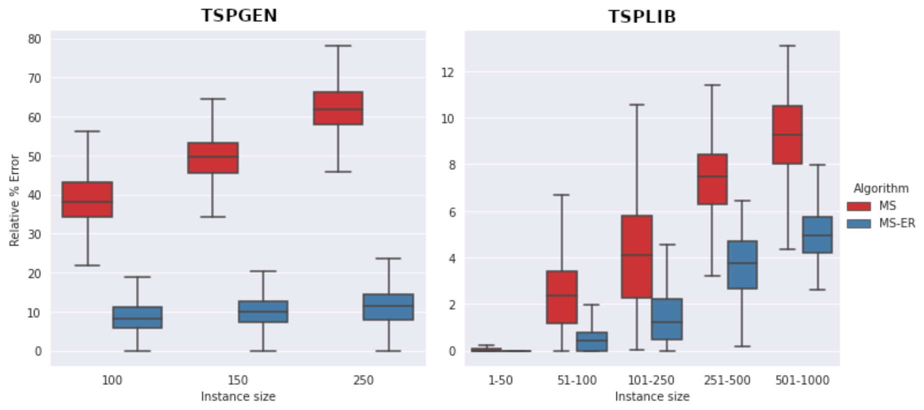


Figure 5.4: Performance comparison between Multi-Start (MS) and Multi-Start plus Extended Reach (ER-MS) implementations for the TSP. The relative percentage error is obtained with respect to the best known solutions.

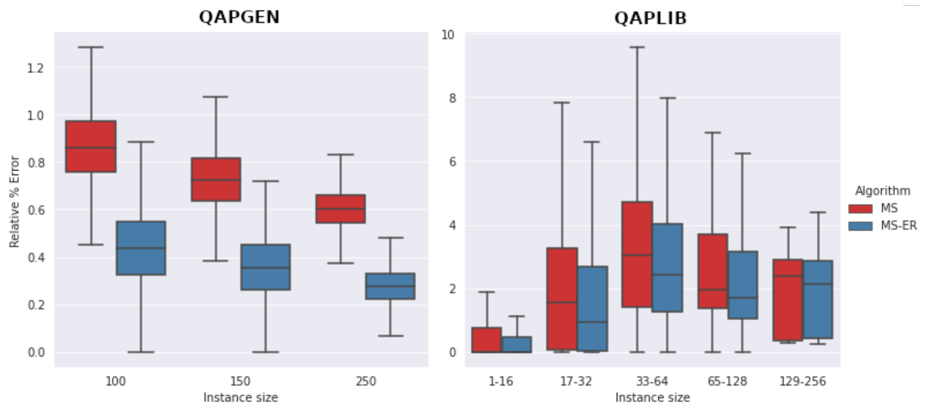


Figure 5.5: Performance comparison between Multi-Start (MS) and Multi-Start plus Extended Reach (ER-MS) implementations for the QAP. The relative percentage error is obtained with respect to the best known solutions.

The conducted experiments reveal that the algorithm adopting the ER method outperforms the Multi-Start heuristic, especially in the randomly generated instances. In the benchmark datasets, particularly in xLOLIB and QAPLIB, due to large number of plateaus present in the instances, the performance gain is not so obvious. However, is worth noting, that MS-ER does exceedingly well in TSPLIB instances, since the adjacency matrices are computed using different distance functions (Euclidean, Manhattan, ...) and the solutions are not prone to ties. The ER method can potentially also escape from plateaus, however, the performance drop in the instances more likely to have them suggests that the method struggles and gets stuck in these optimal regions.

Figure 5.6 also shows the number of local optima encountered on average in every execution. Note that these local optima could be repeated, since the algorithms do not keep a list of the local optima already visited. Nevertheless, MS-ER is able to observe a much larger pool of local optima in every instance.

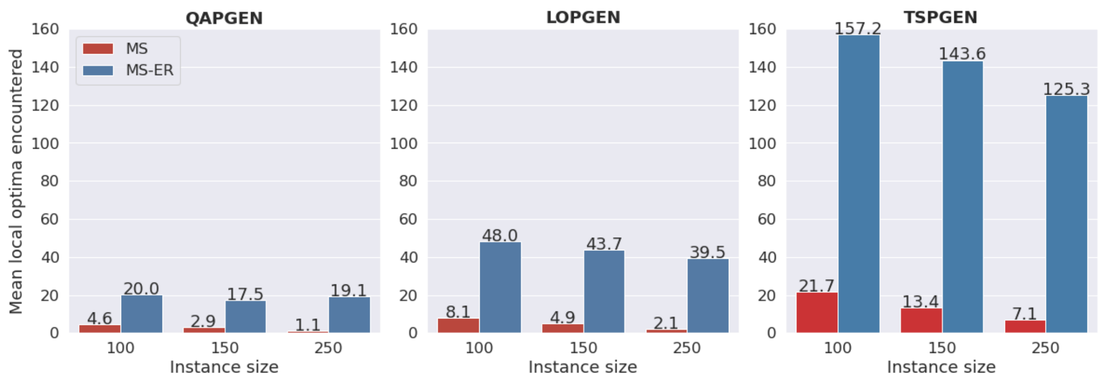


Figure 5.6: Average local optima found per execution in the randomly generated instances.

Finally, the results are summarized in Table 5.2 grouped regarding the combinatorial optimization problem and benchmark. The counters describe the number of instances for which MS-ER obtained better mean objective function value, considering the 20 executions, than the MS algorithm.

	LOP		**TSP**		**QAP**		**Total**
	LOPGEN	xLOLIB	TSPGEN	TSPLIB	QAPGEN	QAPLIB	
MS vs. MS-ER	150	69	150	71	150	89	679
Total Instances	150	78	150	78	150	136	742

Table 5.2: Number of instances where adding the Extended Reach method to the Multi-Start implementations performs better on average than without over 20 executions.

The summary confirms the superior performance of MS-ER, beating the MS algorithm in all randomly generated instances. Regarding the benchmarks libraries, MS-ER obtains better results in 71 out the 78 (∼91%) instances in the TSPLIB dataset, 69 out of 78 (∼88%) in xLOLIB, and 89 out of 136 (∼65%) in QAPLIB.

Chapter 6

Conclusions and Future Work

In this project, the “Extended Reach” method is presented, a novel approach to escape from local optima. Inspired by recently discovered insights in the field of combinatorial landscapes, this new procedure also considers the neighbors of neighbors of the local optima it encounters as candidate solutions to proceed with the search.

The ER method has been adapted and optimized for three widely known combinatorial optimization problems: the Linear Ordering Problem, the Traveling Salesman Problem and the Quadratic Assignment Problem. To that end, the structure of the problems and their most popular neighborhood systems were carefully studied to discover new properties that could improve the efficiency of the developed implementations. In that regard, considering the 2-*opt* neighborhood for the TSP, it has been proven that some second degree neighbors will never be better than the local optima they derive from. Furthermore, in this case for the LOP, a set of formulas based on *insert* operations have been developed to efficiently evaluate neighboring solutions.

The research carried out in this work proved, also for the LOP, that a local optimum of the *insert* neighborhood will always be a local optimum of the 2-*exchange* neighborhood, and not necessarily the other way around. This breakthrough could give a theoretical basis concerning why the *insert* neighborhood gives better results than the 2-*exchange* in regards to the LOP. In addition, it also shows that any Variable Neighborhood Search algorithm for the LOP that switches from the *insert* neighborhood to the 2-*exchange* after finding a local optimum in the first is needlessly spending some of its computational resources.

For the sake of validating the presented approach, an experimental study was conducted where the implementations adopting the ER method clearly showed better performance than the famous Multi-Start meta-heuristic. From the observed results, the ER method successfully escapes from local optima and presents itself as a promising tool for future state-of-the-art local search algorithms. Nevertheless, the experimentation also concludes that the proposed method struggles with instances whose solutions are more likely to have ties and are prone to the formation of plateaus.

All things considered, future versions of the proposed method could benefit from techniques to escape from these plateaus, as well as more advanced and flexible stopping conditions, which instead of just visiting the k best neighbors of the local optima also consider the properties of the optimization problem, the neighborhood system and the instance itself to make an informed decision on when to stop evaluating the second degree neighbors. Further research is required to test the viability of the ER method in a wider range of optimization problems and see how it compares to current state-of-the-art algorithms.

Bibliography

[1] Rainer E Burkard, Stefan E Karisch, and Franz Rendl. "QAPLIB–a quadratic assignment problem library". In: *Journal of Global optimization* 10.4 (1997), pp. 391–403.

[2] Alan M. Frieze et al. "Algorithms for assignment problems on an array processor". In: *Parallel computing* 11.2 (1989), pp. 151–162.

[3] Leticia Hernando, Alexander Mendiburu, and Jose A Lozano. "Anatomy of the attraction basins: Breaking with the intuition". In: *Evolutionary computation* 27.3 (2019), pp. 435–466.

[4] Leticia Hernando, Alexander Mendiburu, and Jose A Lozano. "Journey to the center of the linear ordering problem". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 2020, pp. 201–209.

[5] Rafael Martí et al. "Multi-start methods". In: *Handbook of heuristics*. Springer, 2018, pp. 155–175.

[6] Gerhard Reinelt. "TSPLIB—A traveling salesman problem library". In: *ORSA journal on computing* 3.4 (1991), pp. 376–384.

[7] Valentino Santucci and Josu Ceberio. "Using pairwise precedences for solving the linear ordering problem". In: *Applied Soft Computing* 87 (2020), p. 105998.

[8] Tommaso Schiavinotto and Thomas Stützle. "The linear ordering problem: Instances, search space analysis and algorithms". In: *Journal of Mathematical Modelling and Algorithms* 3.4 (2004), pp. 367–402.

[9] Thomas Stützle. "Iterated local search for the quadratic assignment problem". In: *European journal of operational research* 174.3 (2006), pp. 1519–1539.