Optimization through Iterative Smooth Morphological Transformations

Valentino Santucci*
valentino.santucci@unistrapg.it
University for Foreigners of Perugia
Perugia, Italy

Marco Baioletti*
marco.baioletti@unipg.it
University of Perugia
Perugia, Italy

Marco Tomassini*
marco.tomassini@unil.ch
University of Lausanne
Lausanne, Switzerland

ABSTRACT

In this paper, we introduce SMorph, a new methodology for combinatorial optimization that works in the instance space of the problem at hand. Indeed, given the problem instance to solve, SMorph builds a simplified instance whose optimum is easy to locate, then it iteratively evolves this instance towards the target one by alternating two steps: optimization and smooth transformation of the current instance. The knowledge acquired in each iteration is transferred to next one, while the entire process is designed with the aim of improving the last optimization step. Although the abstract search scheme of SMorph is general enough to be instantiated for a variety of combinatorial optimization problems, here we present an implementation for the well-known Linear Ordering Problem (LOP). Experiments have been conducted on a set of commonly adopted benchmark instances of the LOP, and the results validate the proposed approach.

KEYWORDS

Combinatorial Optimization, Instance Space, Iterative Smooth Morphing Transformation

ACM Reference Format:
Valentino Santucci, Marco Baioletti, and Marco Tomassini. 2024. Optimization through Iterative Smooth Morphological Transformations. In . ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Combinatorial Optimization Problems (COPs) are a class of challenging computational problems that involve finding the best possible solution among a vast number of feasible solutions. COPs arise in various real-world scenarios, such as resource allocation [15], scheduling [8], routing [10], and network design [7]. The quest to efficiently solve these problems has led to the development of powerful algorithms and a variety of theoretical results [22].

However, the vast majority of the COPs of practical interest are NP-hard, so heuristic and metaheuristic approaches have been increasingly proposed in order to overcome the theoretical limitations in finding the optimal solution [6, 18]. These heuristic and metaheuristic approaches, while not guaranteeing optimality, excel in finding good enough solutions within a reasonable time frame, making them indispensable tools for addressing complex COPs in practical applications. Among them, the most popular and prominent examples are: algorithms based on local search techniques (such as [16] and [12]), evolutionary algorithms (such as [20] and [23]), swarm intelligence algorithms (such as [9] and [24]), and model-based gradient search schemes (such as [5] and [21]).

Here, we propose a general heuristic method, called SMorph, for tackling COPs. Given a COP and an instance to solve, SMorph builds a succession of instances such that: the first one is easy to solve to optimality, the last one is the target instance, and the i-th instance is somehow contained in the $(i+1)$-th instance of the succession. Then, a metaheuristic algorithm, such that the output of one execution can be fed as input to another execution, is used to iteratively solve all the instances in the succession.

In order to get advantage from this process, we need to impose two conditions: (i) the succession of intermediate instances has to be generated in such a way that the transition from the initial instance to the target instance is smooth enough in terms of morphological transformations of the fitness landscape navigated by the optimization algorithm, and (ii) the information transferred by the metaheuristic algorithm from one execution to the next should be meaningful for the whole optimization procedure.

Though being a general and abstract scheme which can be used to handle a variety of COPs, to illustrate the main concepts, we provide a concrete implementation of SMorph for the Linear Ordering Problem (LOP) [4, 26]. Moreover, We are also paving the way for a variety of different research avenues, of which this paper should be seen as a preliminary work. In fact, it is important to clearly state from the beginning that our goal here is not at all that of beating specialized algorithms for the LOP problem or any other problem. Since our approach is general, and although it does offer decent performances, it cannot compete with the best algorithms. However, it is in our view a useful algorithm for understanding the relationships between problem instances and their fitness landscapes.

To the best of our knowledge, SMorph is a novel approach for combinatorial optimization, but there are some previous works that can be seen as being correlated to some extent to the ideas herein. In particular, it is conceptually partly inspired by the Adiabatic Quantum Computing (AQC) [1] approach in which an initial spin-glass Hamiltonian in its known ground state is slowly evolved into a final Hamiltonian whose ground state contains the solution to the problem.

There are also some commonalities with dynamic programming and divide-et-impera approaches to optimization, which break down the original problem into smaller and simpler sub-problems. Some similarities can also be found with the CMSA approach [3]

*The authors contributed equally to this research.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

which iteratively solves a series of sub-instances obtained from the tackled problem instance.

The rest of the paper is organized as follows. An abstract and general scheme of SMorph is introduced in Sect. 2. Sect. 3 describes the LOP, while Sect. 4 introduces a concrete implementation of SMorph for the LOP. Experiments on commonly adopted benchmark instances for the LOP are described and discussed in Sect. 5, while conclusion and future lines of research are depicted in Sect. 6.

2 ABSTRACT SCHEME OF SMORPH

In this section we introduce the general scheme of SMorph that, in principle, can be defined and implemented for any COP.

First of all, we denote a COP as a triple $(\mathbb{I}, \mathbb{S}, f)$ where: $\mathbb{I}$ is the set of the instances, $\mathbb{S}$ is the set of feasible solutions (or solution space), and $f : \mathbb{S} \times \mathbb{I} \rightarrow \mathbb{R}$ is the objective function to optimize. As an example, in the well-known Traveling Salesman Problem (TSP): instances are distance matrices among a set of cities, solutions are tours among all the cities, while the objective function is the sum of the distances between adjacent cities in a tour.

Moreover, we are interested in trajectory-based iterative optimization methods, i.e., those algorithms of the form $\mathcal{A} : \mathbb{I} \times \mathbb{S} \rightarrow \mathbb{S}$ that, by taking as inputs both a problem instance $I \in \mathbb{I}$ and a seed solution $x_0 \in \mathbb{S}$, return a solution $x^* \in \mathbb{S}$ whose objective value $f(x^*; I)$ is hopefully optimal or, anyway, good enough. Typical examples of such a kind of algorithms are the Iterated Local Search [16] and the Tabu Search [12].

Given a COP $(\mathbb{I}, \mathbb{S}, f)$ and a particular target instance $I \in \mathbb{I}$, the SMorph method is a meta-scheme that uses a trajectory-based iterative optimization algorithm $\mathcal{A}$ to find the solution $x^* \in \mathbb{S}$ whose objective value $f(x^*; I)$ is hopefully optimal. The pseudocode of SMorph is depicted in Alg. 1.

Algorithm 1 Abstract scheme of SMorph

```
1: I_0 ← P(I)                          ▷ Generate an easy to solve initial instance
2: x_0 ← optimal or good enough solution of I_0
3: k ← 0
4: while I_k ≠ I do                    ▷ Loop until the current instance is the target one
5:     k ← k + 1
6:     I_k ← T(I_{k−1}, x_{k−1}, I)    ▷ Transform the current instance into a new one
7:     x_k ← A(I_k, x_{k−1})           ▷ Optimize the new instance
8: end while
9: return x_k                          ▷ Return the solution found for the target instance
```

By observing Alg. 1, it is possible to identify two crucial components of SMorph: the *preparation function* $\mathcal{P}$ and the *transformation function* $\mathcal{T}$. The function $\mathcal{P}$ generates an initial instance I_0 by simplifying the target instance I (line 1) in a way that makes computationally easy to obtain its optimal solution (line 2). Then, the iterative loop in lines 4–8 transforms the current instance I_k gradually bringing it closer to the target instance I (line 6) and, before advancing to the next iteration, it optimizes the updated instance by executing the algorithm $\mathcal{A}$ seeded with the optimal solution from the previous iteration (line 7). The loop terminates as soon as the current instance matches the target one (line 4). This means that the last execution of $\mathcal{A}$ was run on the target instance I. Therefore, the last obtained solution is returned in line 9. Depending on how $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{T}$ are implemented, the returned solution may not necessarily be optimal, but hopefully sufficiently good, for the target instance I.

The rationale of SMorph is to alternate between a morphological transformation of the search landscape and the execution of an optimization algorithm. Intuitively, the morphological transformation should be guided towards the target landscape and be sufficiently smooth to ensure that the solution found in iteration $k-1$ serves as a good starting point for the optimization step in iteration k. Under these hypotheses, if an optimal solution for the initial instance is known, the whole process should converge towards a good solution for the target instance.

Below, we formalize the properties that $\mathcal{P}$ and $\mathcal{T}$ must satisfy in order to ensure the correctness of the SMorph scheme. Then, the section is closed by some final observations.

2.1 Preparation function

The preparation function has the form $\mathcal{P} : \mathbb{I} \rightarrow \mathbb{I}$ and its purpose is to simplify the given target instance I into an instance $I_0 = \mathcal{P}(I)$ that is easier to solve. While it is possible to first build I_0 and then obtain its optimum (or a good enough solution) by using any optimization algorithm, the opposite process can be more efficient and more effective. Indeed, often it is possible to proceed by first choosing an arbitrary solution $x_0 \in \mathbb{S}$, then construct an instance I_0 related to I and such that x_0 is (provably) its global optimum.

As an example, consider a MAX-TSP[1] instance represented by a distance matrix D. After choosing a tour x_0 arbitrarily, it is straightforward to create a new distance matrix D_0 that maintains the same values as in D for the entries corresponding to adjacent cities in x_0, while all the other entries are set to zero. By design, x_0 is a global optimum of the MAX-TSP instance D_0. Indeed, the objective value of x_0 is equivalent to the sum of all the distances in D_0, a clear upper bound for the optimal value of this MAX-TSP instance.

2.2 Transformation function

The purpose of the transformation function $\mathcal{T}$ is to generate a new instance $I_k = \mathcal{T}(I_{k-1}, x_{k-1}, I)$ by taking as inputs: the previous instance I_{k-1}, its solution x_{k-1}, and the target instance I.

We require that: (i) I_k should be closer than I_{k-1} to I; (ii) I_k should be chosen as the smoothest instance, within a region of the space surrounding x_{k-1}, among all the candidate instances.

The first property can be formalized by relying on a *distance function* d associated to the instance space $\mathbb{I}$. Usually, there exist natural choices for such a distance. As an example, MAX-TSP instances are matrices, thus any distance induced by a matrix p-norm can be considered for the case.

Given I, I_{k-1}, and $I_k = \mathcal{T}(I_{k-1}, x_{k-1}, I)$, on the basis of the chosen distance d, we require that

$$d(I_k, I) = d(I_{k-1}, I) - \delta_k, \text{ with } \delta_k > 0. \quad (1)$$

Moreover, by also indicating with I_N the last instance produced in the transformation loop, it is required that $d(I_N, I) = 0$. In other terms, by denoting with $d_k = d(I_k, I)$, the requirement is that $d_0, d_1, \ldots, d_N$ is a decreasing succession of numbers, with $d_N = 0$. The step-length δ_k needs to be chosen in the design of $\mathcal{T}$. A simple

[1] MAX-TSP is the maximization variant of the TSP (which is a minimization problem). In this example we use MAX-TSP in place of the more common TSP just to simplify the mathematical notation. Indeed, the same findings can be adapted to the TSP as well.

choice is to first decide the total number of iterations N, then set $\delta_k = d(I_0, I)/N$, for $k = 1, \ldots, N$.

The second property is related to the smoothness of the new instance, calculated on a region of the space surrounding the solution found in the previous iteration of SMorph. In order to formalize this property we can consider a *smoothness function* $s : \mathbb{I} \times \mathbb{S} \rightarrow \mathbb{R}$ that, by taking as inputs a candidate instance $J \in \mathbb{I}$ and a solution $x \in \mathbb{S}$, returns a number $s(J, x)$ that measures how smooth J is in the vicinity of x. A possible choice for s is to consider the *random walk correlation function* defined as in [28], but having the foresight to start the random walks from x. This example is probably not efficient enough for practical scenarios, so a problem-dependent smoothness function may be more appropriate. Moreover, it is important to note that, for s to be well-defined, a neighborhood relation on the solution space $\mathbb{S}$ is necessary. However, this is quite natural for many COPs. For instance, in the MAX-TSP, the well-known 2-OPT neighborhood [13] can be considered for the case.

Therefore, given I, I_{k-1}, and $I_k = \mathcal{T}(I_{k-1}, x_{k-1}, I)$, on the basis of the chosen smoothness function s, we require that

$$I_k = \arg\max_{J \in \mathbb{J}} \; s(J, x_{k-1}), \tag{2}$$

where $\mathbb{J} = \{J \in \mathbb{I} : d(J, I) = d(I_{k-1}, I) - \delta_k\}$, i.e., it is the subset of instances satisfying the distance property given in Eq. (1).

In summary, the definition of a concrete transformation function requires the specification of a distance function for the instance space, a step-length, and a smoothness function. The distance function and the step-length are used to identify a candidate set of instances, while the smoothness function allows to select the smoothest instance among the candidates.

2.3 Observations

The effectiveness (i.e., the quality of returned solution) and the efficiency (i.e., the computational effort required) of SMorph depends on how $\mathcal{P}$ and $\mathcal{T}$ are designed.

For the preparation function, it is evident that the closer is the initial instance to the target one, the better it is, but more challenging will be to obtain its global optimum. However, it is worthwhile to note that, in the cases where the initial instance can be derived after arbitrarily choosing its true global optimum (as in the MAX-TSP example provided before), it is always preferable to choose a solution which is as close as possible to the global optimum of the target instance. For instance, in the MAX-TSP example, it is easy to see that an initial instance is closer to the target one, in terms of Euclidean distance, when the solution used to generate such instance has a larger objective value. Therefore, one possibility is to generate the initial solution – then, the initial instance – by executing a constructive heuristic on the target instance.

Regarding the transformation function, its design impacts both the efficiency and effectiveness of SMorph. The larger are the step-lengths (i.e., the δ_k values), the fewer iterations in the main loop of Alg. 1 may be required. On the other hand, smaller step-lengths are likely to produce smoother instances in the candidate set of Eq. (2), thus resulting in smoother morphological transformations of the instances. Clearly, a smooth transformation helps the next optimization step because: the smoother is the transformation, the more likely it is that the optimum of the new instance is close to the optimum of the previous instance, i.e., the starting point for the optimization step. In summary, larger step-lengths improve the efficiency of SMorph but penalize its effectiveness, and vice versa. Therefore, a trade-off between effectiveness and efficiency is required in practical scenarios.

Furthermore, let also note that the smoothness function used by the transformation procedure may be computationally expensive to calculate and maximize. Practically, computationally cheap heuristic procedures for the smoothness may be preferred. We will follow this line later on in the paper when presenting a concrete implementation of SMorph for the LOP.

3 THE LINEAR ORDERING PROBLEM

Consider a $(n \times n)$-matrix $H = [h_{ij}]_{n \times n}$, where $n \geq 2$. The goal of the LOP is to simultaneously reorder the rows and columns of H in such a way that its upper-triangular sum is maximized. To be more precise, the LOP seeks to find, among the set $\mathcal{S}_n$ of all the permutations of $\{1, 2, \ldots, n\}$, the optimal permutation $\sigma^* \in \mathcal{S}_n$ that maximizes the objective function $f(\sigma; H)$, defined as:

$$f(\sigma; H) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} h_{\sigma(i),\sigma(j)}. \tag{3}$$

Any LOP instance H can be normalized by replacing each entry h_{ij} with the value $h_{ij} - \min\{h_{ij}, h_{ji}\}$. This normalization ensures that all entries of H become non-negative and it does not change the order induced by Eq. (3) among the permutations in $\mathcal{S}_n$. For the remainder of this paper we will assume that all LOP instances are normalized, a practice consistent with commonly adopted benchmark suites [18].

In graph theory, if we regard H as the weight matrix of an arc-weighted digraph G, then the LOP is equivalent to determining the maximum-weight tournament graph contained within G. The LOP has been considered in [27] and [14] for natural language processing tasks, while a recent application in the field of machine learning has been proposed in [19]. Further applications in social choice theory, economics, anthropology, and sports are described in [18].

The LOP has been proven to be NP-hard in the seminal work of Garey and Johnson [11]. Since then, many metaheuristic approaches have been proposed. According to [18], the most successful ones are those based on local search procedures adopting the insertion neighborhood. In this context, two permutations σ and π are considered neighbors if and only if π can be obtained by shifting an item from a position i to another position j in σ, and vice versa. Moreover, in [26] it is shown that, given the objective value of σ, it is possible to compute the objective values of all the $(n-1)^2$ neighbors of σ with $\Theta(n^2)$ time steps.

To the best of our knowledge, the state-of-the-art metaheuristics for the LOP are: the Iterated Local Search (ILS) methods proposed in [26] and [4], the memetic algorithm introduced in [17], the very recent CL-TLBO method proposed in [2], and the CD-RVNS algorithm presented in [25]. The latter work also introduced a novel constructive heuristic, namely C-LOP, and experimentally shown its superiority over the previously considered best constructive methods for the LOP.

4 SMORPH FOR THE LINEAR ORDERING PROBLEM

By taking into account the general and abstract scheme of SMorph, as introduced in Sect. 2, here we describe a concrete implementation for the LOP.

SMorph for the LOP works as depicted in Alg. 1 by considering that: the instance space is the set of $n \times n$ matrices, the solution space is the set of n-length permutations $\mathcal{S}_n$, while the objective function is as in Eq. (3).

A concrete implementation of SMorph requires to design the three algorithmic components $\mathcal{A}$ (i.e., the iterative optimization algorithm), $\mathcal{P}$ (i.e., the preparation function) and $\mathcal{T}$ (i.e., the transformation function) for the problem at hand. Their implementations for the LOP are described here below.

4.1 Optimization algorithm

For $\mathcal{A}$, we consider the ILS algorithm based on the insertion neighborhood, that, as described in Sect. 3, is one of the most effective metaheuristic for the LOP.

The ILS used in this study works as follows. It takes a seed solution as input, runs a local search to find the corresponding local optimum, and then iteratively applies three steps: (i) perturbation of the current local optimum, (ii) local search starting from the perturbed solution, (iii) replacement of the current local optimum with the new one if it is better.

The perturbation phase applies 7 random interchanges between pairs of permutation items (a value that has been experimentally calibrated in [26] on a set of commonly adopted benchmark instances), while the local search uses the insertion neighborhood and the *best improvement* strategy, i.e., each neighborhood scan yields the best neighboring solution in terms of objective value.

The number of iterations n_{it} is left as a parameter of the ILS, which, after n_{it} iterations returns the last and best local optimum encountered.

4.2 Preparation function

For the LOP, we devise a preparation function $\mathcal{P}$ which is very similar to the one described in the example provided in Sect. 2.

$\mathcal{P}$ takes as inputs both the target instance $H = [h_{ij}]_{n\times n}$ and a seed solution $\sigma \in \mathcal{S}_n$, then it returns a simplified instance $A = [a_{ij}]_{n\times n}$, which is constructed as follows:

$$a_{\sigma(i),\sigma(j)} = \begin{cases} h_{\sigma(i),\sigma(j)} & \text{if } i < j, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

It is easy to see that, when the rows and columns of A are permuted by σ, all the positive entries appear in the upper triangular part of the permuted matrix. This implies that $f(\sigma; A)$ is equal to the sum of all the entries of A, i.e., a clear upper bound for the LOP. Therefore, the permutation σ is, by construction, a global optimum for the instance A.

We also devise two strategies for choosing the seed solution σ: SMorph-R and SMorph-H, which select σ, respectively, at random and by applying the constructive heuristic C-LOP (see Sect. 3) to the target instance H. Consequently, the computational cost for generating the initial instance and its global optimum is: $\Theta(n^2)$ for SMorph-R, evident from Eq. (4), and $\Theta(n^2 \log n)$ for SMorph-H, determined by the average computational cost of running C-LOP [25].

4.3 Transformation function

To describe the transformation function $\mathcal{T}$ implemented for the LOP, for the sake of presentation, we will use the following notation: $A = [a_{ij}]_{n\times n}$ represents the previous instance in input, whose solution $\sigma \in \mathcal{S}_n$ is given by the previous iteration of SMorph, $H = [h_{ij}]_{n\times n}$ denotes the target instance, while $B = [b_{ij}]_{n\times n}$ will be the returned instance as computed by $\mathcal{T}$, i.e., $B = \mathcal{T}(A, \sigma, H)$.

According to Sect. 2, it is useful to identify: a suitable distance function d, a step-length δ, and an appropriate smoothness function s.

In this preliminary study, we choose the Hamming distance between matrices as d. Thus, for two $(n \times n)$-matrices A and B, $d(A, B)$ counts the number of different entries between A and B. We also choose a constant step-length $\delta = 1$. Therefore, the candidate instances to consider in $\mathcal{T}$ are all the instances that can be generated by moving an entry from the target instance H into instance A, provided that this entry has not already been moved in the previous iterations of SMorph.

We use the notation $C^{(ij)} = \left[c_{kl}^{(ij)}\right]_{n\times n}$ to represent the candidate instance obtained by moving the generic entry (i, j). $C^{(ij)}$ is defined as:

$$c_{kl}^{(ij)} = \begin{cases} a_{kl} & \text{if } k \neq i \text{ and } l \neq j, \\ h_{ij} & \text{otherwise.} \end{cases} \tag{5}$$

Therefore, $\mathbf{C} = \left\{C^{(ij)} : a_{ij} \neq h_{ij} \text{ and } i, j = 1, \ldots, n\right\}$ is the set of candidates, which contains $O(n^2)$ instances.

Within $\mathbf{C}$, $\mathcal{T}$ has to select the smoothest instance. With this regard, we have designed an efficient smoothness function s that, given an instance $C^{(ij)} \in \mathbf{C}$, is defined as follows:

$$\begin{aligned} s\left(C^{(ij)}, \sigma\right) = & -\left|f\left(\sigma; C^{(ij)}\right) - f\left(\sigma^{(ij)}; C^{(ij)}\right)\right| \\ & -\left|f\left(\sigma; C^{(ij)}\right) - f\left(\sigma^{(ji)}; C^{(ij)}\right)\right|, \end{aligned} \tag{6}$$

where: $\sigma^{(ij)}$ is the permutation obtained by shifting the item at position i to position j in σ, while $\sigma^{(ji)}$ is the permutation obtained by shifting the item at position j to position i in σ.

The two absolute differences in the right hand side of Eq. (6) represent the magnitudes of the changes in objective function value when the insertion moves $i \rightarrow j$ and $j \rightarrow i$ are applied to the solution σ. The rationale behind this choice is that the smaller are the changes in objective values between σ and its neighbors, the smoother is the instance in the neighborhood of σ, which, we recall, will be the starting point for the next application of $\mathcal{A}$. Note also that the two minus signs in front of the absolute values are there because the smoothest candidate instance is the one that maximizes s.

For the sake of efficiency, Eq. (6) considers only two neighbors of σ. This limitation is not overly restrictive because, as shown in the analysis of the insertion neighborhood presented in [4], there can be at most $2n$ neighbors of σ whose objective values change when the underlying instance is changed in only one entry (i, j).

Moreover, $\sigma^{(ij)}$ and $\sigma^{(ji)}$ are the two neighbors most directly affected by the change. Hence, having a constant number of neighbors allows the incremental computation for the smoothness of all the candidate instances in $O(n^2)$ time steps in total. The key is to use the value $f(\sigma; A)$, which has already been computed in the previous iteration of SMorph, and to scan the candidate instances by following an appropriate order, similar to what is described in [26] for accelerating local search methods.

In summary, the transformed instance $B = \mathcal{T}(A, \sigma, H)$ is the instance that, among all those sharing the same entries as A except for one taken from H, maximizes the smoothness function defined in Eq. (6), that is:

$$B = \underset{C^{(ij)} \in \mathbf{C}}{\arg\max}\ s\left(C^{(ij)}, \sigma\right). \tag{7}$$

4.4 Observations and computational improvement

Taking into account the algorithmic components $\mathcal{A}$, $\mathcal{P}$, and $\mathcal{T}$ introduced in the previous subsections, the SMorph implementation for the LOP can be outlined through the pseudocode presented in Alg. 2, which closely follows the abstract scheme provided in Alg. 1, with only few differences.

Algorithm 2 SMorph for the LOP

```
1:  A, σ ← P(H)                                   ▷ P is defined in Sect. 4.2
2:  N ← d(A, H)                                   ▷ d is defined in Sect. 4.3
3:  for k ← 1 to N do
4:      B ← T(A, σ, H)                            ▷ T is defined in Sect. 4.3
5:      if (i, j) is the entry where B differs from A, and σ^-1(i) > σ^-1(j) then
6:          σ ← A(B, σ)                           ▷ A is defined in Sect. 4.1
7:      end if
8:      A ← B
9:  end for
10: return σ
```

First of all, it is worth noting that there are two initialization strategies to choose from in line 1, i.e., SMorph-R and SMorph-H, as described in Sect. 4.2.

Furthermore, it is possible to pre-determine the number of iterations, which is equal to the Hamming distance between the initial instance A and the target instance H (see line 2). Since we are dealing with normalized LOP instances and considering that A is initialized with at least $n(n-1)/2$ entries from H, we have that SMorph requires $O(n^2)$ iterations.

As shown in line 5, we can save computational time by not always executing the optimization algorithm $\mathcal{A}$. Indeed, let assume that σ is the global optimum of the previous instance A, and B was selected by $\mathcal{T}$ as the candidate $C^{(ij)}$, i.e., its values match those of A, except for the entry (i, j), taken from H. Then, when considering the matrix B permuted by σ, we have that the new entry (i, j) can go either in the upper or lower triangular part of the permuted matrix. When it goes in the upper triangular part, i.e., when $\sigma^{-1}(i) < \sigma^{-1}(j)$, it is easy to see that: (i) $f(\sigma; B) = f(\sigma; A) + h_{ij}$, and (ii) the objective values of the neighbors of σ either remain unchanged or increase by the same quantity h_{ij}. Consequently, σ is the global optimum also for the new instance B, thus there is no need to run the optimization algorithm on B. Even though we cannot guarantee that σ will always be the global optimum for the current instance at every SMorph iteration, we anyway apply this trick to save computational time.

Finally, it is easy to see that, if the target instance in input is in normal form, then the initial and all the intermediate instances generated by SMorph are in normal form as well.

5 EXPERIMENTS

5.1 Experimental Setup

Computational experiments were conducted with a twofold purpose: to validate the proposed approach, and to investigate its effectiveness.

The experiments were carried out on 218 commonly adopted benchmark instances, divided into the following sets: (i) the *IO* set, which contains 50 real-world economic input-output tables, whose size varies from 44 to 79, and whose optimal values are known; (ii) the *RandB* set, consisting of 90 matrices randomly generated in an asymmetric manner and such that their optimal values are unknown; (iii) the *xLOLIB* set, which can be further divided into two subsets, *xLOLIB_150* and *xLOLIB_250*, each comprising 39 instances of size, respectively, 150 and 250, with unknown optima. All the benchmark suites are publicly available at https://grafo.etsii.urjc.es/optsicom/lolib.html. As said in Sect. 1, because the goal is not to be competitive with the best algorithm for the LOP, we use small and medium-size instances which facilitates the experimentation by cutting on computation time.

For the main goal, i.e., to validate the proposed approach, since the SMorph for the LOP performs multiple executions of the ILS on a succession of slightly different instances, we compared it with a multi-start version of the ILS algorithm (MS-ILS), which performs a series of independent ILS executions on the given target instance. All the runs of the ILS, both in SMorph and MS-ILS, are set to terminate after $n_{it} = 5$ iterations have been performed (see Sect. 4.1). Moreover, to ensure a fair comparison, both SMorph and MS-ILS executions evaluate the same number of solutions. Therefore, if SMorph outperforms MS-ILS, this will be an indication that employing the SMorph gradual process to establish a good starting point for optimizing the target instance is at least better than repeatedly attempting different starting points at random.

For the sake of completeness, we also compared SMorph with Long-ILS, i.e., the ILS algorithm set to run with the same budget of evaluations observed in SMorph executions, without explicitly limiting its number of iterations n_{it}.

Practically, SMorph is first executed 15 times per instance and, for each execution, we recorded the number of evaluations performed. Then, also MS-ILS and Long-ILS are executed 15 times per instance using, as total budget, the average number of evaluations observed in SMorph executions.

Finally, taking into account the two initialization strategies described in Sect. 4.2, we have devised two variants of SMorph, namely SMorph-H and SMorph-R. Furthermore, we apply both heuristic and random initialization also to Long-ILS and all the ILS executions within MS-ILS. Consequently, we will use the acronyms Long-ILS-H, Long-ILS-R, MS-ILS-H, and MS-ILS-R. In summary, we have a total of six algorithms, each of them executed 15 times on each of the 218 benchmark instances under consideration, resulting in a total of 19 620 executions.

5.2 Experimental Results

To facilitate a fair comparison and aggregation of results across different instances, the final objective value v produced by each execution is converted into the *relative percentage deviation* measure, defined as $rpd = 100 \cdot (best - v)/best$, where *best* represents the best objective value observed among all executions of each algorithm on the same instance.

Tab. 1 shows the average *rpd*s for each algorithm, aggregated across individual benchmark sets and across the entire benchmark suite. These results can be commented as follows.

First and foremost, it is important to note that both SMorph variants exhibit better overall *rpd*s than the two MS-ILS variants. This advantage of SMorph over MS-ILS is particularly evident on the larger and more challenging instances, i.e., those in the two xLOLIB sets. In contrast, MS-ILS appears to perform better on the RandB instances, even slightly outperforming Long-ILS, while the IO benchmark set does not clearly distinguish between the competing algorithms.

Second, Long-ILS shows better *rpd*s than SMorph, considering both variants. However, it is interesting to note that, when moving from xLOLIB_150 to xLOLIB_250, while Long-ILS degrades its *rpd*s, SMorph improves its performance.

To further explore these results, we present a boxplot graph in Fig. 1, which illustrates the distribution of *rpd* values achieved by the competing algorithms, aggregated for each benchmark set.

Fig. 1 clearly confirms the previous observations. Furthermore, it allows to see that SMorph is able to achieve very good peak results, comparable with those of Long-ILS. However, it also reveals that SMorph lacks the robustness exhibited by the other competitors.

Lastly, we complete our analysis by presenting in Tab. 2 the win-tie-loss comparisons involving both SMorph variants. For the sake of presentation, we focus on the most challenging benchmark sets, i.e., xLOLIB_150 and xLOLIB_250. The results of each comparison, between two algorithms "A_1 vs A_2", are expressed as a triple $w/t/l$, indicating that: A_1 outperformed A_2 in w instances, A_1 and A_2 have the same performance in t instances, and A_1 was outperformed by A_2 in l instances. Two types of comparisons are presented: `MWU`, which employs the Mann Whitney U test (with a confidence level of 0.05) to compare results from all executions of both algorithms on a particular instance, and `Best`, which directly compares the best objective values recorded by the algorithms in all executions on a specific instance.

Tab. 2 consolidates the previous observations. In particular, it shows that SMorph-H is consistently better than MS-ILS-H, both in terms of average and best results. The same is true for SMorph-R when compared to both MS-ILS-R and MS-ILS-H. Regarding the comparison between SMorph variants and Long-ILS variants, while the latter show better results overall, it is interesting to note that SMorph-H achieved statistically indistinguishable performance compared to Long-ILS-H in 17 out of 39 instances of the xLOLIB_250 set. Furthermore, in terms of peak performance, SMorph-H was more effective than Long-ILS-H in 9 out of 39 instances of the xLOLIB_250 set. Finally, regarding the SMorph-H vs SMorph-R comparison, let observe that, though the heuristic initialization is almost always preferable in terms of average performance, SMorph-R was able to achieve better peak performance than SMorph-H in 35 out of 78 xLOLIB instances.

In conclusion, based on the results presented, we can confidently assert that SMorph has indeed achieved greater effectiveness compared to MS-ILS, thus meaning that the gradual process of SMorph, which transitions an initial instance towards a target instance, allows to produce a seed solution for the last ILS execution that is consistently better than selecting it randomly or heuristically, even after multiple repetitions. On the other hand, the effectiveness of SMorph is not yet at the level of Long-ILS which is an important part of the best algorithms for LOP.

6 CONCLUSION AND FUTURE WORK

We have introduced a novel approach, called SMorph, for understanding the structural changes that take place in the instance space when the algorithm smoothly progresses towards the optimum. This method draws partial inspiration from the Adiabatic Quantum Computing approach, but unlike the latter, it is firmly grounded in classical computing environments.

SMorph relies on three key components: an iterative optimization algorithm, a preparation function, and a transformation function. The preparation function simplifies the given target instance, making its solution easier to find. Subsequently, the instance undergoes iterative refinement and is gradually transformed into the target instance. During each iteration, the iterative optimization algorithm is executed on the current instance, using the previous solution as starting point. The rationale behind this approach is that a smooth morphological transformation of a problem instance should result in a new problem instance, whose solution is closely related to the previous one. Therefore, SMorph seeks to apply this principle by iteratively guiding the process from an easy initial instance towards the given target instance.

In addition to presenting an abstract definition of SMorph, we have also introduced a concrete implementation for the Linear Ordering Problem (LOP). Computational experiments were conducted on a set of commonly used benchmark instances for the LOP. The results clearly validate the proposed approach but indicate that improvements are needed if SMorph is to be used as an efficient optimization algorithm in the future.

The findings discussed in this paper offer opportunities for further investigation from various perspectives.

One of the most interesting ideas for further study is to conduct a series of fitness landscape analyses on the sequence of instances generated by SMorph. This will enhance our comprehension of the algorithm's dynamics and provide valuable insights for its future development.

Additionally, SMorph as an optimization algorithm can be extended in each one of its three key components along several directions, although we do not yet know if it will achieve performances close to those of the best algorithms. Notably, the most promising ones are: (i) exploring other distance measures for the instance space, like e.g. the Euclidean or Manhattan distance; (ii) examining different strategies for determining the step-length of transformations at each iteration, also to limit the total number of iterations

Benchmark set	SMorph-H	SMorph-R	MS-ILS-H	MS-ILS-R	Long-ILS-H	Long-ILS-R
IO	0.01	0.01	0.00	0.00	0.00	0.00
RandB	0.11	0.13	0.02	0.01	0.02	0.03
xLOLIB_150	0.44	0.53	0.70	0.87	0.18	0.19
xLOLIB_250	0.39	0.52	0.88	1.09	0.24	0.25
Overall	0.20	0.25	0.29	0.35	0.08	0.09

Table 1: Average relative percentage deviations (*rpd*s).

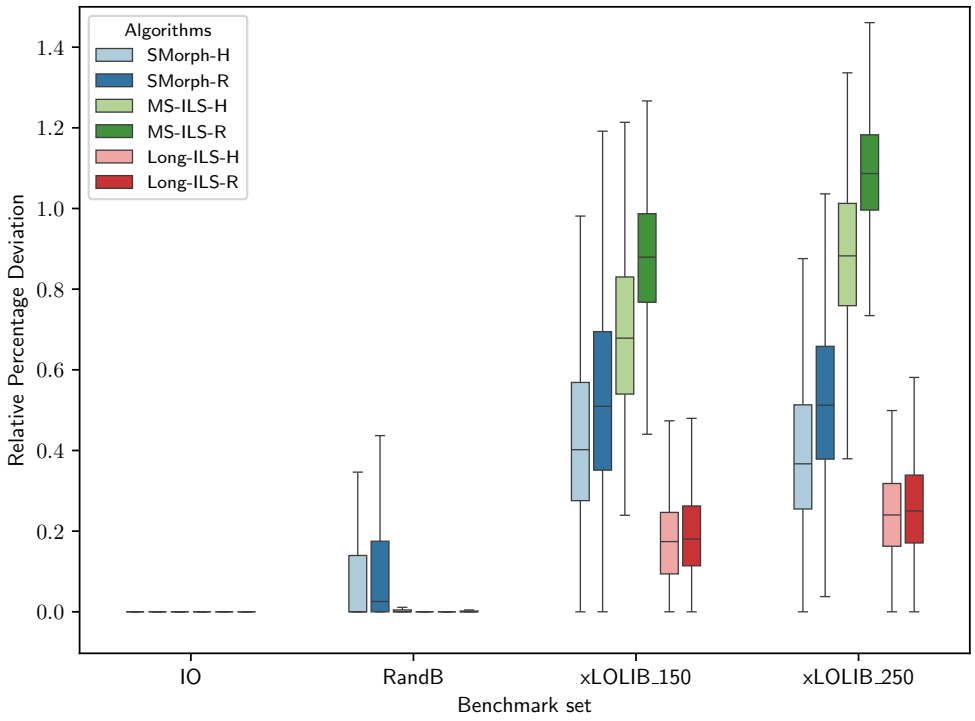

Figure 1: Boxplot of the relative percentage deviations recorded in all the executions.

Type	Algorithms	xLOLIB_150	xLOLIB_250
MWU	SMorph-H vs SMorph-R	13 / 25 / 1	22 / 15 / 2
	SMorph-H vs MS-ILS-H	37 / 2 / 0	39 / 0 / 0
	SMorph-H vs Long-ILS-H	0 / 5 / 34	0 / 17 / 22
	SMorph-R vs MS-ILS-R	35 / 4 / 0	39 / 0 / 0
	SMorph-R vs MS-ILS-H	23 / 13 / 3	37 / 1 / 1
	SMorph-R vs Long-ILS-R	0 / 0 / 39	0 / 2 / 37
Best	SMorph-H vs SMorph-R	16 / 0 / 23	27 / 0 / 12
	SMorph-H vs MS-ILS-H	39 / 0 / 0	39 / 0 / 0
	SMorph-H vs Long-ILS-H	3 / 0 / 36	9 / 0 / 30
	SMorph-R vs MS-ILS-R	39 / 0 / 0	39 / 0 / 0
	SMorph-R vs MS-ILS-H	39 / 0 / 0	39 / 0 / 0
	SMorph-R vs Long-ILS-R	2 / 0 / 37	2 / 0 / 37

Table 2: Win-Tie-Loss comparisons of SMorph vs all the other algorithms.

performed by SMorph; (iii) studying different methodologies for selecting candidate instances and evaluating their smoothness within the transformation function; (iv) using optimization algorithms other than the ILS scheme, such as population-based algorithms.

Another avenue for extension is the application of SMorph to other combinatorial optimization problems, encompassing various types of search spaces, including binary string spaces. Moreover, the methodology for generating the initial instance can be further extended and investigated to produce benchmark instances with known global optima, and, from a theoretical point-of-view, it is worth investigating the pace of the entire iterative process to ensure that the global optimum of the new instance lies within the neighborhood, or in the same basin of attraction, of the global optimum of the previous one.

ACKNOWLEDGMENTS

Valentino Santucci has been partially supported by the research projects: "Università per Stranieri di Perugia – Finanziamento Dipartimentale alla Ricerca per Progetti di Ricerca di Ateneo – FDR 2023", "Università per Stranieri di Perugia – Finanziamento Dipartimentale alla Ricerca per Progetti di Ricerca di Ateneo – FDR 2024".

REFERENCES

[1] Tameem Albash and Daniel A Lidar. 2018. Adiabatic quantum computation. *Reviews of Modern Physics* 90, 1 (2018), 015002.
[2] Abdelkamel Ben Ali. 2024. Comprehensive learning TLBO with recursive precedence-based solution construction and multilevel local search for the linear ordering problem. *Expert Systems with Applications* 238 (2024), 122315.
[3] Christian Blum, Pedro Pinacho, Manuel López-Ibáñez, and José A Lozano. 2016. Construct, merge, solve & adapt a new general algorithm for combinatorial optimization. *Computers & Operations Research* 68 (2016), 75–88.
[4] Josu Ceberio, Alexander Mendiburu, and Jose A Lozano. 2015. The linear ordering problem revisited. *European Journal of Operational Research* 241, 3 (2015), 686–696.
[5] Josu Ceberio and Valentino Santucci. 2023. Model-based Gradient Search for Permutation Problems. *ACM Transactions on Evolutionary Learning and Optimization* 3, 4 (2023), 1–35.
[6] Bastien Chopard and Marco Tomassini. 2018. *An introduction to metaheuristics for optimization.* Springer.
[7] Maria da Conceição Cunha and Joaquim Sousa. 1999. Water distribution network design optimization: simulated annealing approach. *Journal of water resources planning and management* 125, 4 (1999), 215–221.
[8] Lawrence Davis. 2014. Job shop scheduling with genetic algorithms. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications.* Psychology Press, 136–140.
[9] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. 2006. Ant colony optimization. *IEEE computational intelligence magazine* 1, 4 (2006), 28–39.
[10] Burak Eksioglu, Arif Volkan Vural, and Arnold Reisman. 2009. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering* 57, 4 (2009), 1472–1483.
[11] M. R. Garey and D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)* (first edition ed.). W. H. Freeman.
[12] Fred Glover and Manuel Laguna. 1998. *Tabu search.* Springer.
[13] David S Johnson. 1990. Local optimization and the traveling salesman problem. In *International colloquium on automata, languages, and programming.* Springer, 446–461.
[14] Allen Kim and Steven Skiena. 2022. Chapter Ordering in Novels. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing.* 3838–3848.
[15] Eliane Maria Loiola, Nair Maria Maia De Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. 2007. A survey for the quadratic assignment problem. *European journal of operational research* 176, 2 (2007), 657–690.
[16] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. 2003. Iterated local search. In *Handbook of metaheuristics.* Springer, 320–353.
[17] Lázaro Lugo, Carlos Segura, and Gara Miranda. 2022. A diversity-aware memetic algorithm for the linear ordering Problem. *Memetic Computing* 14, 4 (2022), 395–409.
[18] Rafael Marti and Gerhard Reinelt. 2022. *Exact and Heuristic Methods in Combinatorial Optimization.* Vol. 175. Springer.
[19] Nitin Kumar Mishra and Pramod Kumar Singh. 2022. Linear ordering problem based classifier chain using genetic algorithm for multi-label classification. *Applied Soft Computing* 117 (2022), 108395.
[20] Yuichi Nagata and Shigenobu Kobayashi. 2013. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing* 25, 2 (2013), 346–363.
[21] Yann Ollivier, Ludovic Arnold, Anne Auger, and N. Hansen. 2017. Information-geometric optimization algorithms: A unifying picture via invariance principles. *Journal of Machine Learning Research* 18, 18 (2017), 1–65.
[22] Christos H Papadimitriou and Kenneth Steiglitz. 1998. *Combinatorial optimization: algorithms and complexity.* Courier Corporation.
[23] Valentino Santucci, Marco Baioletti, and Gabriele Di Bari. 2021. An improved memetic algebraic differential evolution for solving the multidimensional two-way number partitioning problem. *Expert Systems with Applications* 178 (2021), 114938.
[24] Valentino Santucci, Marco Baioletti, and A. Milani. 2019. Tackling permutation-based optimization problems with an algebraic particle swarm optimization algorithm. *Fundamenta Informaticae* 167, 1-2 (2019), 133–158.
[25] Valentino Santucci and Josu Ceberio. 2020. Using pairwise precedences for solving the linear ordering problem. *Applied Soft Computing* 87 (2020), 105998.
[26] Tommaso Schiavinotto and Thomas Stützle. 2004. The linear ordering problem: Instances, search space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms* 3 (2004), 367–402.
[27] Roy Tromble and Jason Eisner. 2009. Learning linear ordering problems for better translation. In *Proceedings of the 2009 conference on empirical methods in natural language processing.* 1007–1016.
[28] Vesselin K Vassilev, Terence C Fogarty, and Julian F Miller. 2003. Smoothness, ruggedness and neutrality of fitness landscapes: from theory to application. *Advances in evolutionary computing: theory and applications* (2003), 3–44.