A Review of Metrics on Permutations for Search Landscape Analysis

Tommaso Schiavinotto and Thomas Stützle
Darmstadt University of Technology
Computer Science Department, Intellectics Group
Hochschulstr. 10, 64289 Darmstadt, Germany.
tommaso.schiavinotto@gmail.com; stuetzle@informatik.tu-darmstadt.de

Abstract

Search landscape analysis has become a central tool for analysing the dependency of the performance of stochastic local search algorithms on structural aspects of the spaces being searched. Central to search landscape analysis is the notion of distance between candidate solutions. This distance depends on some underlying basic operator and it is defined as the minimum number of operations that need to be applied to one candidate solution for transforming it into another one. For operations on candidate solutions that are represented by permutations, in almost all researches on search landscape analysis surrogate distance measures are applied, although efficient algorithms exist in many cases for computing the exact distances. This discrepancy is probably due to the fact that these efficient algorithms are not very widely known. In this article, we review algorithms for computing distances on permutations for the most widely applied operators and present simulation results that compare the exact distances to commonly used approximations.

Keywords: Search landscape analysis, metrics, distance, permutations.

1 Introduction

It is widely acknowledged that a careful analysis of the search landscape in which stochastic local search algorithms such as tabu search, simulated annealing, evolutionary algorithms or iterated local search navigate is central to a deeper understanding of the behaviour of these algorithms. In fact, search landscape analysis (often also called fitness landscape analysis) is very widely used in the evolutionary computation community [20, 21, 23, 25, 27] and, more in general, in stochastic local search [18, 30, 31, 34, 35, 41].

Of central importance to search landscape analysis is the concept of distance between solutions. Given two solutions s and s', their distance $d(s, s')$ is typically defined as the minimal number of applications of a certain *basic operation* in order to transform the first solution into the second one. Depending on the solution representation and the basic operation involved, the distance can easily be computed. This is the case, for example, if we consider a binary representation like for the satisfiability problem in propositional logic, and the basic operation of *flipping* bits. In that case, the well-known *Hamming distance*, which can be computed in linear time as a function of the bit-string length, corresponds to the desired distance. In other cases, in particular, when solutions are represented as permutations—a representation that naturally arises in many sequencing problems—in virtually any such endeavours, the authors used approximations to the exact distances [26, 30, 36, 39]. This is done, although for several, frequently used basic operations the exact distances can be computed efficiently; in fact, these results are mainly taken from permutation theory [10]. Nevertheless, for some other standard operators the problem of whether the exact distances can be computed in polynomial time is still open; in these cases, however, often good approximations are available, and most of the relevant work on this topic is done in computational biology for genome rearrangement problems. A comprehensive review of this research can be found in Christie's PhD thesis [11], while some special cases are tackled in the PhD thesis of Vergara [38].

In this article, we review known results on the efficient computation of distance functions and we present a simulation study that compares the most frequently used approximations to the exact distances. The article is structured as follows. In the next section, we define the concepts on landscapes used in the remainder of the article, while Section 3 gives the necessary background on permutations and examples of the most commonly used basic operations and the corresponding distances. Section 4 is dedicated to an experimental study of the quality of the approximations that are most widely used for linear permutations and we conclude in Section 5.

2 Neighbourhoods, landscapes and distances

Search landscape analysis is a tool used for better understanding the interdependence between algorithms, structure of the search spaces and performance. To define a search landscape, several concepts are important. First, one needs to define the search space $\mathcal{S}$ of candidate solutions an algorithm is searching. Second, in local search algorithms we need to define a *neighbourhood* between candidate solutions:

Definition 1 (Neighbourhood)
A neighbourhood is a mapping $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$, that associates to each solution a set of candidate solutions, called neighbours.

The set of neighbours of s is called $\mathcal{N}(s)$ and a usual requirement is that a solution s is not in $\mathcal{N}(s)$. Usually, a neighbourhood is not defined explicitly, but through an operator. In that case, the neighbourhood of a candidate solution is defined to be the set of candidate solutions that can be reached by applying the operator once.

Definition 2 (Operator)
An operator Δ is a collection of operator functions $\delta : \mathcal{S} \to \mathcal{S}$ such that:

$$s' \in \mathcal{N}(s) \iff \exists \delta \in \Delta . \delta(s) = s'.$$

As usual, it can be avoided that $s \in \mathcal{N}(s)$ by forcing that $\delta(s) \neq s$ for all $s \in \mathcal{S}$. The application of an *operator function* we also call *move*.

Search space and neighbourhood define the *neighbourhood graph* $\mathcal{G}_{\mathcal{N}}(\mathcal{S}, E_{\mathcal{N}})$, in which the set of nodes corresponds to the set of candidate solutions and the set of edges is defined as $E_{\mathcal{N}} = \big\{\langle s, s'\rangle \big| s, s' \in \mathcal{S} \wedge s' \in \mathcal{N}(s)\big\} = \big\{\langle s, s'\rangle \big| s, s' \in \mathcal{S} \wedge \exists \delta \in \Delta . s' = \delta(s)\big\}$, that is, two candidate solutions s, s' are connected by an edge if s' is in the neighbourhood of s. All the operators that we investigate in the following result in connected and symmetric neighbourhood graphs.

Furthermore, if $s, s' \in \mathcal{S}$ we denote with

$$\Phi(s, s') = \big\{(s_1, \ldots, s_h) | s_1 = s, s_h = s', \forall i : 1 \leq i \leq h - 1 . \langle s_i, s_{i+1}\rangle \in E_{\mathcal{N}}\big\}$$

the set of paths in $\mathcal{G}_{\mathcal{N}}$ that start at s and end in s'. A path indicates a sequence of moves for transforming s into s'. If $\phi = (\phi_1, \ldots, \phi_h) \in \Phi(s, s')$ we indicate with $|\phi|$ the length $|\phi| = h$ of the path.

Given the neighbourhood graph, we can introduce the notion of *distance* between solutions.

Definition 3 (Distance)
We define the distance $d_{\mathcal{N}}$ between two solutions $s, s' \in \mathcal{S}$ to be the length of the shortest path between s and s' in $\mathcal{G}_{\mathcal{N}}$:

$$d_{\mathcal{N}}(s, s') = \min_{\phi \in \Phi(s,s')} |\phi|.$$

Finally, we can define the search landscape by additionally associating to each candidate solution in a neighbourhood graph a value that rates its quality. This value is defined through an *evaluation function*.

Definition 4 (Search landscape)
Given a search space $\mathcal{S}$, a neighbourhood function $\mathcal{N}$, and an evaluation function $f : \mathcal{S} \to \mathbb{R}$, the search landscape *is a triple $\mathcal{L} = \langle \mathcal{S}, \mathcal{N}, f\rangle$.*

The notion of search landscape and search landscape analysis can be traced back to theoretical works on evolutionary systems [42] and it is a frequently applied tool in the research on stochastic local search algorithms [18]. The notion of distance plays a significant role in several types of analysis like that of the ruggedness of the search landscapes [41] or fitness-distance correlations [20].

3 Operators and Distances

The distance between solutions depends on the representation of candidate solutions and on the operator. In the following we first introduce necessary details about permutations that are required for understanding the part on operators.

3.1 Permutations

Here, we focus on representations based on (linear) permutations. We indicate by $\Pi(n)$ (or simply Π) the set of all permutations of the numbers $\{1\ 2\ \ldots\ n\}$; the sequence $(1\ 2\ \ldots\ n)$, which is the identity permutation, is indicated by $\imath$. If $\pi \in \Pi(n)$ and $1 \leq i \leq n$, π_i indicates the element at position i of permutation π; $pos_\pi(i)$ is the position at which element i is located in π. A permutation can also be seen as a bijective function, where $\pi(i) = \pi_i$. We can define the permutation product $(\pi \cdot \pi')$ as the composition of such functions:

$$(\pi \cdot \pi')_i = \pi'(\pi(i)).$$

Consider, for example, the two permutations $\pi = (2\ 3\ 4\ 1)$ and $\pi' = (3\ 4\ 2\ 1)$. For π we have $pos_\pi(1) = 4$, $pos_\pi(2) = 1$, $pos_\pi(3) = 2$, and $pos_\pi(4) = 3$. The function composition $\pi \cdot \pi'$ would result in the sequence $(4\ 2\ 1\ 3)$.

The identity permutation corresponds to the identity function. For each permutation π, there exists a permutation π^{-1} such that $\pi^{-1} \cdot \pi = \pi \cdot \pi^{-1} = \imath$. Such a permutation can be obtained as $(\pi^{-1})_i = pos_\pi(i)$, which implies the following property:

$$pos_{\pi^{-1} \cdot \pi'}(i) = pos_{\pi'}(\pi_i). \tag{1}$$

For example, for the permutation $\pi = (2\ 3\ 4\ 1)$ we have that $\pi^{-1} = (4\ 1\ 2\ 3)$. $\pi^{-1} \cdot \pi'$ then would be the permutation $(2\ 3\ 1\ 4)$ and one can easily verify that for this example Equation 1 holds. Permutations as functions form a non-Abelian group with function composition as binary operation. (The binary operation needs to be associative; it must be closed; a neutral and an inverse element w.r.t. the operation must exist.) In fact, permutation theory is an important part of group theory that can be traced back to Cayley, 1849 [10].

Note, that when permutations are considered, the operator functions of a given operator are bijective functions that can be represented as well as permutations ($\delta_\mathcal{N} \in \Pi$). Therefore, the operator is a subset of the permutation set ($\Delta_\mathcal{N} \subset \Pi$, $\imath \notin \Delta_\mathcal{N}$).

For permutations, the distance function has the following property:

$$d_\mathcal{N}(\pi', \pi) = d_\mathcal{N}(\pi^{-1} \cdot \pi', \imath). \tag{2}$$

This can easily be seen as follows. $d_\mathcal{N}(\pi, \pi') = k$ means that, because of what we said before,

$$\exists \pi^1, \ldots, \pi^k \in \Delta_\mathcal{N}.\ \pi' \cdot \pi^1 \cdot\ \cdots\ \cdot \pi^k = \pi,$$

which implies

$$\exists \pi^1, \ldots, \pi^k \in \Delta_\mathcal{N}.\ \pi^{-1} \cdot \pi' \cdot \pi^1 \cdot\ \cdots\ \cdot \pi^k = \pi^{-1} \cdot \pi = \imath.$$

This means that the distance between two permutations π, π' is the smallest number of times an operator is to be applied to sort $\pi^{-1} \cdot \pi'$.

A *cycle* is a specific type of permutation. A permutation $\bar{\pi}$ is a k-cycle (with $k > 1$), if, given k distinct values $(i_1, \ldots, i_k)$, it can be written as

$$\bar{\pi}_j = \begin{cases} j & j \notin i_1, \ldots, i_k \\ i_{h+1} & j = i_h, 1 \leq h < k \\ i_1 & j = i_k \end{cases}$$

A 1-cycle is defined to be the identity $\imath$. A k-cycle is usually represented listing only the characterising values in the right order as $(i_1\ i_2\ \ldots\ i_k)$. Let us give one example for a 4-cycle. In the example, on the

left the permutation is represented using a notation that associates positions (upper row) to elements (lower row) and on the right side is given the 4-cycle.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 4 & 5 & 3 & 2 & 6 & 7 & 8 \end{bmatrix} = (2\ 4\ 3\ 5)$$

Any permutation can be uniquely expressed as a composition of disjoint cycles. This cycle decomposition (*factorisation*) is unique except for the order of the cycles, which can be arbitrary, and the first element of each cycle, which can be an arbitrary element of the cycle. In such a representation of a permutation π, all elements are listed, possibly using 1-cycles if needed. We indicate the number of cycles as $c(\pi)$. An example for a cycle decomposition of a permutation (the same as above with the elements 1 and 8 interchanged) is

$$\pi = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 8 & 4 & 5 & 3 & 2 & 6 & 7 & 1 \end{bmatrix} = (1\ 8)(2\ 4\ 3\ 5)(6)(7)$$

with $c(\pi) = 4$.

Finding the cycle decomposition is a very straightforward task: Given an element π_i, first add (position) i to the cycle. Then, at each step a next position is considered; the next position of the cycle is given by π_i and the process (addition of a position, consideration of the next position) is iterated. If the process returns to the initial position, the construction of the cycle terminates and all the elements that have been met in the construction belong to the same cycle of the factorisation. (Consider, for example, the permutation in the previous example and start with element $\pi_2 = 4$; then, position 4 is considered next and we have $\pi_4 = 3$; this is followed by position $\pi_3 = 5$ and finally again position $\pi_5 = 2$ at which point the construction of the cycle ends. The elements added in this process correspond to the second cycle, (2 4 3 5), of the decomposition.) To obtain all cycles, the next cycle construction starts at a position that was not yet considered; all cycles are found once all positions have been considered.

Finally, for the following we also need the concept of a *transposition*, which in permutation theory is a 2-cycle. Any transposition $\bar{\pi}$ is the inverse of itself ($\bar{\pi} \cdot \bar{\pi} = \imath$) and any permutation can be represented as a composition of transpositions.

3.2 Swap Operator

The swap operator is formally defined as:

$$\Delta_S = \left\{ \delta_S^i \middle| 1 \leq i < n \right\},$$

where $\delta_S^i : \Pi(n) \to \Pi(n)$ and

$$\delta_S^i(\pi_1\ \ldots\ \pi_i\ \pi_{i+1}\ \ldots\ \pi_n) = (\pi_1\ \ldots\ \pi_{i+1}\ \pi_i\ \ldots\ \pi_n).$$

In other words, the swap operator swaps the position of two adjacent elements in a permutation.

For the computation of the distance, recall that *bubble sort* is a sorting algorithm that sorts any permutation using the minimal number of swaps [19, 38]. This fact, together with the property given by Equation 1 on the previous page implies that we can use bubble sort to compute the swap-operator based distance in $O(n^2)$.

Consider now a measure $s : \Pi \to 1, \ldots, n$ of the sortedness of a permutation π defined as

$$s(\pi) = \#\left\{ \langle i, j \rangle \middle| 1 \leq i < j \leq n,\ pos_\pi(j) < pos_\pi(i) \right\}$$

This measure counts how many couples in π have the wrong relative order. A sortedness of 0 means that the permutation is ordered, while the maximum value of $n(n-1)/2$ for this measure indicates that all couples are in the wrong order, which corresponds to a reversal of $\imath$. Each time a swap is done, this measure is decreased by one. Since at the end of bubble sort the sortedness is 0, the sortedness gives also the number of swaps done by *bubble sort* and

$$d_{\mathcal{N}_S}(\pi', \pi) = d_{\mathcal{N}_S}(\pi^{-1} \cdot \pi', \imath) = s(\pi^{-1} \cdot \pi').$$

Interestingly, the sortedness and the number of swaps to be done in bubble sort correspond also to the *precedence based distance metric* [30, 40], which is defined as

$$\#\left\{ \langle i, j \rangle \middle| 1 \leq i < j \leq n,\ \ pos_{\pi'}(\pi_j) < pos_{\pi'}(\pi_i) \right\}.$$

The diameter of the search space for the *swap* operator is $n(n-1)/2$.

3.3 Interchange Operator

The interchange operator is given as follows:

$$\Delta_X = \left\{ \delta_X^{ij} \middle| 1 \leq i < j \leq n \right\},$$

where $\delta_X^{ij} : \Pi(n) \to \Pi(n)$ and

$$\delta_X^{ij}(\pi) = (\pi_1 \ \ldots \ \pi_{i-1} \ \pi_j \ \pi_{i+1} \ \ldots \ \pi_{j-1} \ \pi_i \ \pi_{j+1} \ \ldots \ \pi_n).$$

Actually, this operator corresponds to the set of all transpositions (see Section 3.1). The neighbourhood graph $\mathcal{G}_{\mathcal{X}}$ is a *transposition graph*.

Any permutation can be obtained by a series of transpositions applied to $\imath$ and since Cayley it is known that the minimum number of transpositions needed to obtain a given permutation π from $\imath$ is $n - c(\pi)$. This corresponds to the distance $d_{\mathcal{N}_X}(\imath, \pi)$ and, since the transposition graph is symmetric, it also corresponds to $d_{\mathcal{N}_X}(\pi, \imath)$. Thus, considering two permutations π and π' we have

$$d_{\mathcal{N}_X}(\pi, \pi') = d_{\mathcal{N}_X}(\pi^{-1} \cdot \pi', \imath) = n - c(\pi^{-1} \cdot \pi').$$

The permutation $\pi^{-1} \cdot \pi'$ can be sorted by interchanging adjacent elements in a cycle (the first and the last are considered adjacent), which splits the cycle into two cycles, one with one element and the other one with all the remaining elements. Hence, to transform a permutation to a composition of cycles of single elements—the ordered sequence $\imath$—we need to apply for each cycle c_j of length $\#c_j$ to apply $\#c_j - 1$ transpositions.

In summary, we have the following result. To compute the distance $d_{\mathcal{N}_X}(\pi, \pi')$ between two sequences, it is enough to first compute $\pi^{-1} \cdot \pi'$, which can be done in $O(n)$, and then the number of permutation cycles that compose that permutation, which again can be done in linear time. Thus, we have a linear-time algorithm for computing the required distance.

An algorithm that computes this distance can be given as follows:

$c \leftarrow 0$
$\bar{\pi} = \pi' \cdot \pi^{-1}$
for $i \leftarrow 1 \ldots n$ **do**
  if $\bar{\pi}_j$ *is not checked* **then**
    $c \leftarrow c + 1$
    $j \leftarrow i$
    repeat
      mark $\bar{\pi}_j$ *as checked*
      $j \leftarrow \bar{\pi}_j$
    until j = i
$d_X(\pi, \pi') \leftarrow n - c$

This algorithm generates the disjoint cycles that compose $\pi^{-1} \cdot \pi'$. If these cycles were stored, it would be possible to determine all possible paths that lead from π to π' and vice-versa.

In his PhD thesis [1], Bachelet proposes an algorithm that applies transpositions between elements that are adjacent in a cycle of $\pi^{-1} \cdot \pi'$ without directly computing $\pi^{-1} \cdot \pi'$. However, there was no proof or argument given in [1] that indicated that this algorithm actually computed the exact distance for the interchange operator. The algorithm of Bachelet is as follows.

$d \leftarrow 0$
for $i \leftarrow 1 \ldots n$ **do**
  while $\pi_i \neq \pi'_i$ **do**
    $j \leftarrow pos_\pi(\pi'_i)$
    $\pi' \leftarrow \delta_X^{ij}(\pi')$
    $d \leftarrow d + 1$
  end while
$d_X(\pi, \pi') \leftarrow d$

The diameter of $\mathcal{G}_{\mathcal{N}_X}$ is $n - 1$.

3.4 The Insert operator

The third operator, *insert*, is defined as

$$\Delta_I = \left\{ \delta_I^{ij} \middle| 1 \leq i \leq n, 1 \leq j \leq n,\ j \neq i \right\},$$

where $\delta_I^{ij} : \Pi \to \Pi$ and

$$\delta_I^{ij}(\pi) = \begin{cases} (\pi_1\ \ldots\ \pi_{i-1}\ \pi_{i+1}\ \ldots\ \pi_j\ \pi_i\ \pi_{j+1}\ \ldots\ \pi_n) & i < j \\ (\pi_1\ \ldots\ \pi_j\ \pi_i\ \pi_{j+1}\ \ldots\ \pi_{i-1}\ \pi_{i+1}\ \ldots\ \pi_n) & i > j \end{cases}$$

With this operator, the distance $d_{\mathcal{N}_I}(s, s')$ is known as *Ulam's metric* [6, 13]:

$$d_{\mathcal{N}_I} = n - |lis(\pi^{-1} \cdot \pi')|,$$

where $lis(\pi)$ is the *longest increasing subsequence* (LIS) of the permutation π and $|\cdot|$ denotes the length of the given sequence. An increasing subsequence is formed by the elements of π that are in increasing order in the given sequence. Note that more than one LIS may exist. Intuitively, the LIS indicates which elements are already in the *right order* and need not to be moved for sorting the sequence. For example, we have four maximal LISs in the following permutation:

$$(1\ 3\ 4\ 10\ 9\ 6\ 2\ 5\ 8\ 7) \xrightarrow{lis} \begin{matrix} (1\ 3\ 4\ 5\ 8) \\ (1\ 3\ 4\ 5\ 7) \\ (1\ 3\ 4\ 6\ 8) \\ (1\ 3\ 4\ 6\ 7) \end{matrix}.$$

In this case, five insert moves would be needed to sort the sequence: if we use the first LIS, we would need to put into the right order the elements $10, 9, 6, 2, 7$. The order in which the elements are moved can be arbitrary. Trivially, it is always possible to increase the length of the longest increasing subsequence with one *insert* move. Furthermore, there is no way to increase the length of an LIS by two and the only sequence with an LIS of length n is the sorted one.

The algorithm given next can be used to compute the length of the LIS for a permutation π. This algorithm uses four operations on a data structure, which stores the elements of the permutation that already have been examined. The four operations are:

- *insert(v)*: Insert v in the data structure;
- *delete(v)*: Delete v from the data structure;
- *prev(v)*: The largest element smaller than v in the data structure.
- *next(v)*: The smallest element larger than v in the data structure.

The algorithm scans a permutation π from left to right and for each element checked it stores the LIS in which an element participates, which is the length of the partial LIS that uses this element. In the algorithm, $L[u]$ gives the length of the LIS that ends in u and, at termination, l is the length of the LIS for π.

```
l ← 0
for i ← 1...n do
    insert(π_i)
    if prev(π_i) ≠ nil then
        L[π_i] ← L[prev(π_i)] + 1
    else
        L[π_i] ← 1
    if L[π_i] > l then
        l ← L[π_i]
    if (prev(π_i) ≠ nil) ∧ (L[next(π_i)] = L[π_i]) then
        delete(next(π_i))
```

For the efficiency of the algorithm it is important that the four operations can be performed as quickly as possible. A straightforward implementation using some kind of balanced binary search tree (such as AVL or Red-Black trees [12]), achieves an overall complexity of the algorithm of $O(n\ log(n))$. A smarter implementation using the van Emde Boas data structure [37] of the same algorithm will obtain a complexity of $O(n\ log(log(n)))$. In [5] it is shown, how all LISs can be enumerated. All the permutations of all LISs of $\pi^{-1} \cdot \pi'$ correspond to all possible paths on $\mathcal{G}_{\mathcal{N}_I}$ to reach π' from π and vice-versa. Another algorithm for finding an LIS in a sequence is described in [29]; this algorithm has a higher complexity of $O(n\ log(n))$ when compared to the best version of the previous one, but it allows for an easier implementation. This algorithm is also used in [32].

The diameter of $\mathcal{G}_{\mathcal{N}_I}$ is $n - 1$.

3.5 Other operators

There are several other operators that are widely applied to permutations. An example is the well-known *2-edge-exchange* operator used in the Travelling Salesman Problem (TSP), which replaces two edges of a tour with two different edges so that the feasibility of the solution is maintained. 2-edge-exchange corresponds to a *reversal* of a subsequence in a circular permutation.

It has been shown that sorting a linear permutation by *reversals* (this problem is known as *sorting by reversal*—SBR) is an $\mathcal{NP}$-hard problem [8]. Additionally, it was shown that SBR on linear and circular permutations is linearly equivalent [33] and that there does not exist any polynomial-time algorithm that provides an approximation ratio of 1.0008 [4]. Currently, the best known approximation guarantee is of 1.375 [3]. In many researches on the search landscape structure of the TSP, the bond distance has been used [7, 24, 27, 36]: given two TSP tours, the bond distance counts the number of edges that are not in common among the two tours. This metric is similar to the adjacency metric defined in Section 4 but applied to circular permutations. The bond distance gives a 2-approximation of the exact distance by reversal between tours, much worse than the 1.375 ratio of the best known approximation algorithm; however, it has the advantage that it is straightforward to compute in $\mathcal{O}(n)$.

Caprara proposed a branch-and-price algorithm for SBR on linear permutations that achieves fairly good results [9]; for example, it requires a few (around 500 seconds on a Digital Ultimate Workstation 500MHz) for instances of the size of 200 [9], but few seconds for size 100. In order to use this algorithm for circular permutations, however, all $2n$ possible linearisations would need to be sorted [33], which makes the exact distance computation already computationally prohibitive for a few hundreds of points. Completely different is the case for signed permutations; in fact, in this case there exists polynomial algorithms to find the sorting by reversal [2, 22].

A second, commonly used operator consists in moves of a contiguous subsequence into another position in the sequence. In local search, this operation is often referred to as *block-moves*; in computational biology, this operation is called *transposition*, which raises some confusion with the meaning of this term in permutation theory (see Section 3.1). Sorting by block-moves has not been studied as extensively as SBR. In fact, there is no proof that this problem is $\mathcal{NP}$-hard; however, there also does not exist a proven polynomial-time algorithm for solving it. A block-move applied to circular permutations corresponds to the well-known 3-edge-exchange move in the TSP. In [15], it was shown that sorting by block-moves over linear or circular permutations are linearly equivalent problems. In the same paper, the best approximation algorithm known to-date is given, which gives an approximation guarantee of 1.5 times the exact solution; the algorithm has a complexity of $\mathcal{O}(n^{\frac{3}{2}}\sqrt{\log n})$.

Some work has been done on special cases of this problem, in particular for sorting by bounded block-moves, where the length of the block moved is bounded [16] and for sorting by short block-moves, where the block moved is of maximum length three [17]. The latter is of particular interest because it corresponds to the *Or-exchange* moves [28]; the authors of [17] conjecture that a polynomial-time algorithm exists for solving this problem; however, presently such an algorithm is not known.

4 Metrics used as Approximation

Fitness-distance correlation analysis is often applied using a surrogate distance metric that (hopefully) gives a good approximation to the true distance [26, 30, 36, 39]. Probably this is the case because the authors were not aware that the exact distance can easily be computed. In this section, we present an experimental study about the correlation between the distances computed by exact algorithms and approximations using four different surrogate metrics for linear permutations. The four approximations

are the precedence metric, which, as shown in Section 3.2, measures the exact distance when using the swap operator, and the following three.

Permutation Hamming Distance The permutation Hamming distance $\delta_H(\pi, \pi')$ is used, for example, for the Quadratic Assignment Problem [26, 36]. It counts the number of positions at which two permutations mismatch and, hence, the distance values range from 0 (both are the same) to n (the length of the sequences); more formally we have

$$\delta_H(\pi, \pi') = \#\{i | \pi_i \neq \pi'_i\}.$$

Adjacency based distance The adjacency relation of two elements in a permutation π is defined as:

$$adj_\pi(u, v) \equiv |pos_\pi(u) - pos_\pi(v)| = 1.$$

Based on the adjacency relation, the adjacency-based distance $\delta_A(\pi, \pi')$ between two permutations π and π' can be defined as:

$$\delta_A(\pi, \pi') = n - 1 - \#\{1 \leq i < n | adj_{\pi'}(\pi_i, \pi_{i+1})\}.$$

This measure counts the number of elements that are adjacent in one permutation but not in the other. This metric was used, for example, in [30].

Position based distance Given two permutations π, π' the position-based distance $\delta_P(\pi, \pi')$ measures the sum of the differences between the positions of the elements. More formally, it is measured as

$$\delta_P(\pi, \pi') = \sum_{i=1}^{n} |pos_\pi(i) - pos_{\pi'}(i)|.$$

This metric is considered, for example, in [30, 39].

Since the surrogate metrics are used as approximations to the true distances, it is interesting to analyse how good this approximation is. To do so, we generated 16 000 random permutations of length $n = 100$ and measured the distance to the identity permutation $\imath$ for all surrogate metrics and exact metrics. When doing so, we faced one important problem. If we generate random permutations and when considering one specific, fixed metric, the distances measured for this metric are distributed with a fixed mode and very small variance; that is, all random permutations were all closely around one specific distance value. However, for our study we aimed at having the distances distributed over all possible values and this for all metrics. To generate such permutations, we started from the identity permutation and we iteratively applied random moves and recorded all permutations generated after each random move. Here, we considered four different moves: *insert*, *interchange*, *reverse* and a hybrid one that chooses uniformly at random among the previous three. For each of these possibilities, random moves were applied 200 times and for each the process was repeated 10 times. To gain further diversification, for every generated sequence the reversed sequence is added. This results in a total of $4 \cdot 200 \cdot 10 \cdot 2 = 16\,000$ permutations.

For each permutation we computed the distance to $\imath$ using the distance measures for which we know a polynomial time algorithm and the approximations. (Recall that the value for the exact swap distance is the same as the one given by the precedence metric.) In this way, we can do pairwise comparisons of how well the correlation is between any pair of distance measures; this was done by computing pairwise linear correlations and additionally using pairwise scatter-plots. Figure 1 shows in the lower left triangle the scatter plots, while in the upper right triangle the empirical correlations are given. The values for all the distances are normalised so that they range between 0 and 1.

As can be seen from the plots, the highest correlation of 0.96 is between swap and the position-based distance measure. For several other pairs, the correlation coefficients are still reasonably high, as it is the case between *interchange* moves and the Hamming distance. However, this approximation has several problems: a lot of sequences result to be at the maximum distance from the identity permutation according to the Hamming distance, considering instead the real distance, they are much closer. Why this discrepancy may appear can be explained as follows. Consider the sequence $(2\ 1\ 4\ 3\ \ldots\ n\ n-1)$—in the case n is even—where each couple of elements i and $i+1$ (with i even) are swapped; this sequence can be sorted with $n/2$ interchange moves, but the sequence differs in all positions from the identity sequence, resulting in a Hamming distance of n. The approximation is better when the value of the real

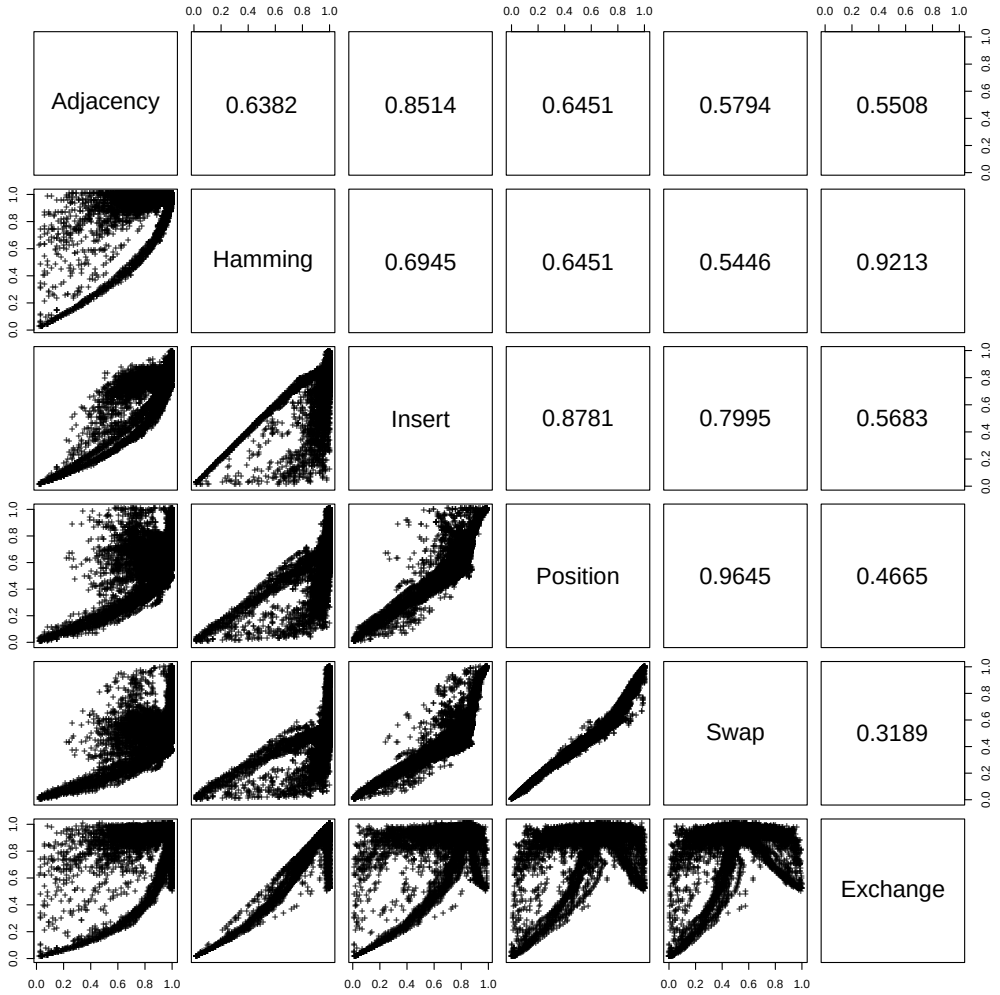

Figure 1: Given are scatter plots for the pairwise relationship between distance metrics for permutations (lower triangle) and the corresponding correlation coefficients (upper triangle). Recall that `Swap` measures the same value as the precedence metric. See the text for details on the generation of the permutations.

distance is actually small. This effect is due to the fact that the Hamming distance is a 2-approximation of the real one.

If we consider instead the *insert* move, there is no metric that can effectively approximate the real distance. The adjacency based distance is a bad choice for all the basic moves we considered in this analysis. As we said, a similar distance for circular permutations (called *bound* distance) is instead extensively used in the study of the TSP search space.

5 Conclusions

In this paper we review metrics for permutations that seem to have been ignored in search landscape analysis. In particular, we focused on three basic operators in problems represented by linear permutations for which fast algorithms are known to compute the exact distances. When we introduced the exact distance for the operators *insert*, *interchange* and *swap*, we also explained how the moves needed for transforming a permutation into another one can easily be obtained. This is interesting for techniques like *Path Relinking* [14], but may also be incorporated into the design of specialised crossover operators (like the distance-preserving crossover [24]) in evolutionary algorithms to interpolate between two parents.

Besides the review of the metrics, we experimentally studied the tightness of the most widely used approximations to the exact distances. In general, these approximations seem not to be very good and,

even worse, they do not give any advantage from the computational point of view. This suggests that it could be worthwhile to carefully re-consider previously achieved results in light of the true distance metrics and to use the exact distances in future endeavours of this kind.

Another issue arises for metrics that cannot be computed efficiently. However, at least for some cases, approximation algorithms with known bounds on the deviation from the exact distance are known. It would be interesting to develop SLS algorithms for obtaining distances close to the exact ones and then to conduct a correlation study, similar to that of Section 4, between the approximations of the exact distances and the usually used surrogate distance metrics for these cases.

Acknowledgment

This work was supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] V. Bachelet. *Métaheuristiques Parallèles Hybrides: Application au Problème D'affectation Quadratique.* PhD thesis, Université des Sciences et Technologies de Lille, 1999.

[2] P. Berman and S. Hannenhalli. Fast sorting by reversal. In D. S. Hirschberg and E. W. Myers, editors, *Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching*, number 1075 in Lecture Notes in Computer Science, pages 168–185. Springer Verlag, Berlin, Germany, 1996.

[3] P. Berman, S. Hannenhalli, and M. Karpinski. 1.375–approximation algorithm for sorting by reversals. In *10th European Symposium on Algorithms (ESA)*, Lecture Notes in Computer Science, pages 200–210. Springer Verlag, Berlin, Germany, 2002.

[4] P. Berman and M. Karpinski. On some tighter inapproximability results (extended abstract). In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, volume 1644, pages 200–209. Springer Verlag, Berlin, Germany, 1999.

[5] S. Bespamyatnikh and M. Segal. Enumerating longest increasing subsequences and patience sorting. *Information Processing Letters*, 76(1–2):7–11, 2000.

[6] W. A. Beyer, M. L. Stein, and S. M. Ulam. Metric in biology, an introduction. Preprint LA-4937, University of California, Los Alamos, 1972.

[7] K. D. Boese, A. B. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimization. *Operations Research Letters*, 16(2):101–113, 1994.

[8] A. Caprara. Sorting by reversals is difficult. In M. Waterman S. Istrail, P. Pevzner, editor, *Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB'97)*, pages 75–83. ACM press, New York, NY, USA, 1997.

[9] A. Caprara, G. Lancia, and S.-K. Ng. Sorting permutations by reversals through branch-and-price. *INFORMS Journal on Computing*, 13(3):224–244, 2001.

[10] A. Cayley. Note on the theory of permutations. *Philosophical Magazine*, 34:527–529, 1849.

[11] D. A. Christie. *Genome Rearrangement Problems*. PhD thesis, Univerisity of Glasgow, 1998.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, second edition, 2001.

[13] D. Critchlow. Ulam's metric. In *Encyclopedia of Statistical Sciences*, volume 9, pages 379–380. John Wiley & Sons, 1988.

[14] F. Glover. A template for scatter search and path relinking. In J. Hao, E. Lutton, E. Ronald, Shoenauer M., and D. Snyers, editors, *Artificial Evolution 1997*, volume 1363 of *Lecture Notes in Computer Science*, pages 13–54, Berlin, 1997. Springer Verlag, Berlin, Germany.

[15] T. Hartman and R. Shamir. A simpler 1.5-approximation algorithm for sorting by transpositions. In R. A. Baeza-Yates, E. Chávez, and M. Crochemore, editors, *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching*, volume 2676 of *Lecture Notes in Computer Science*, pages 156–169. Springer Verlag, Berlin, Germany, 2003.

[16] L. S. Heath and J.-P. C. Vergara. Sorting by bounded block-moves. *Discrete Applied Mathematics*, 88:181–206, 1998.

[17] L. S. Heath and J.-P. C. Vergara. Sorting by short block-moves. *Algorithmica*, 28(3):323–354, 2000.

[18] H. H. Hoos and T. Stützle. *Stochastic Local Search—Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2004.

[19] M. R. Jerrum. The complexity of finding minimum-length generator sequences. *Theoretical Computer Science*, 36:265–289, 1985.

[20] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1995.

[21] L. Kallel, B. Naudts, and C. R. Reeves. Theoretical aspects of evolutionary computing. In L. Kallel, B. Naudts, and A. Rogers, editors, *Properties of Fitness Functions and Search Landscapes*, pages 175–206. Springer Verlag, Berlin, Germany, 2001.

[22] H. Kaplan, R. Shamir, and R. E. Tarjan. Faster and simpler algorithm for sorting signed permutations by reversals. In *RECOMB '97: Proceedings of the First Annual International Conference on Computational Molecular Biology*, page 163. ACM press, New York, NY, USA, 1997.

[23] S. A. Kauffman. *The Origins of Order*. Oxford University Press, New York, NY, USA, 1993.

[24] P. Merz. *Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies*. PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany, 2000, 2000.

[25] P. Merz and B. Freisleben. Fitness landscapes and memetic algorithm design. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 244–260. McGraw Hill, London, UK, 1999.

[26] P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4(4):337–352, 2000.

[27] H. Mühlenbein. Evolution in time and space – the parallel genetic algorithm. In G. J. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1991.

[28] I. Or. *Traveling Salesman-type Problems and their Relation to the Logistics of Regional Blood Banking*. PhD thesis, Northwestern University, Department of Industrial Engineering and Management Science, Evanston, IL, USA, 1976.

[29] M. Orlowski and M. Pachter. An algorithm for the determination of a longest increasing subsequence in a sequence. *Computers & Mathematics with Applications*, 17(7):1073–1075, 1989.

[30] C. R. Reeves. Landscapes, operators and heuristic search. *Annals of Operations Research*, 86:473–490, 1999.

[31] C. M. Reidys and P. F. Stadler. Combinatorial landscapes. *SIAM Review*, 44(2):3–54, 2002.

[32] J. Scharnow, K. Tinnefeld, and I. Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, 3(4):349–366, 2004.

[33] A. Solomon, P. Sutcliffe, and R. Lister. Sorting circular permutations by reversal. In F. K. H. A. Dehne, J-R Sack, and M. H. M. Smid, editors, *Proceedings of the 8th International Workshop on Algorithms and Data Structures*, volume 2748, pages 75–83. Springer Verlag, Berlin, Germany, 2003.

[34] P. F. Stadler. Towards a theory of landscapes. In R. Lopéz-Peña, R. Capovilla, R. García-Pelayo, H. Waelbroeck, and F. Zertuche, editors, *Complex Systems and Binary Networks*, volume 461 of *Lecture Notes in Physics*, pages 77–163. Springer Verlag, Berlin, Germany, 1995.

[35] P. F. Stadler. Fitness landscapes. In M. Lässig and A. Valleriani, editors, *Biological Evolution and Statistical Physics*, pages 187–207. Springer Verlag, Berlin, Germany, 2002.

[36] T. Stützle and H. H. Hoos. $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[37] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6:80–82, 1977.

[38] J.-P. C. Vergara. *Sorting by Bounded Permutations*. PhD thesis, Virginia Tech, 1997.

[39] J. P. Watson, L. Barbulescu, L. D. Whitley, and A. E. Howe. Contrasting structured and random permutation flow-shop scheduling problems: Search space topology and algorithm performance. *INFORMS Journal on Computing*, 14(2):98–123, 2002.

[40] J.-P. Watson, J. C. Beck, A. E. Howe, and L. D. Withley. Problem difficulty for tabu search in job-shop scheduling. *Artificial Intelligence*, 143:189–217, 2003.

[41] E. D. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63(5):325–336, 1990.

[42] S. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings of the Sixth Congress on Genetics*, page 365, 1932.