J Heuristics (2007) 13:35–47
DOI 10.1007/s10732-006-9001-3

Distance measures based on the edit distance for permutation-type representations

Kenneth Sörensen

Received: 10 May 2005 / Revised: 22 March 2006 / Accepted:
22 March 2006 / Published online: 19 December 2006

Abstract In this paper, we discuss distance measures for a number of different combinatorial optimization problems of which the solutions are best represented as permutations of items, sometimes composed of several permutation (sub)sets. The problems discussed include single-machine and multiple-machine scheduling problems, the traveling salesman problem, vehicle routing problems, and many others. Each of these problems requires a different distance measure that takes the specific properties of the representation into account. The distance measures discussed in this paper are based on a general distance measure for string comparison called the edit distance. We introduce several extensions to the simple edit distance, that can be used when a solution cannot be represented as a simple permutation, and develop algorithms to calculate them efficiently.

Keywords Distance measures · Edit distance · Permutation-type representations

Recent research points out that accurate distance measures between solutions of optimization problems can significantly contribute to the design of metaheuristics for these problems. The development of effective distance measures however is not a trivial task. A good distance measure should provide an accurate estimate of the difference — or similarity — between two given solutions to an optimization problem.

The use of distance measures in the design of evolutionary algorithms has been proposed for maintaining a diverse population (Mauldin, 1984) or to locate a set of different solutions to a multi-modal problem such as in crowding (Mahfoud, 1992) or fitness sharing (Goldberg and Richardson, 1987). Other metaheuristics such as scatter search (Glover et al., 2000a,b) or memetic algorithms with population management (Sörensen and Sevaux, 2006) use distance measures primarily for diversification purposes, i.e. to guide the search into previously unexplored regions of the search space.

K. Sörensen (✉)
University of Antwerp, Faculty of Applied Economics, Prinsstraat 13, B–2000 Antwerp, Belgium
e-mail: kenneth.sorensen@ua.ac.be

Greistorfer and Voß (2005) discuss the effects of diversification and solution quality in population-based metaheuristics. In multi-objective optimization, the number of non-dominated solutions may be very large, and researchers recognize the fact that the distance measures can be used to present the decision maker with a small set of diverse solutions.

Of course, different solution representations require different distance measures. For binary representations, the Hamming distance is the most natural distance measure to use. For solutions that are best represented as vectors of real numbers, a variation of the Minkowsky-r-distance (Euclidean, Manhattan, Chebychev, ...) can be used.

However, for problems of which the solutions are most naturally represented as a permutation of a set of items (i.e. problem attributes), there is no agreed-upon "natural" distance measure. Such problems are characterized by the fact that the *order* in which the items appear in the solution is important. We refer to this class of representations as *permutation-type representations*. Many solutions to problems in combinatorial optimization can be represented in this way, including solutions to scheduling problems, routing problems, and many other.

A large number of distance measures for permutation-type representations exist in the literature. Ronald (1998) defines the exact match and the deviation distance functions. The *exact match distance* is an extension of the Hamming distance and considers the exact position of an item in the solution to be important. Given two permutations $p = (p_1, p_2, \ldots, p_n)$ and $q = (q_1, q_2, \ldots, q_n)$, this distance is defined as follows.

$$\begin{aligned} d_m(p,q) &= n - \sum_{i=1}^{n} m_i(p,q) \\ \text{where} \quad m_i(p,q) &= \begin{cases} 1 & \text{if } p_i = q_i \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{1}$$

The *deviation distance* uses the positional deviation between matching item values to calculate the distance between permutations. This distance measure is essentially the same as the rank correlation measure known in statistics as "Spearman's footrule" (Kendall and Gibbons, 1990), except that the latter is scaled between -1 and 1. Assuming without loss of generality that both p and q are permutations of the natural numbers $[1, \ldots, n]$, this distance measure is defined as

$$d_d(p,q) = \sum_{i=1}^{n} \frac{|j-k|}{n-1} \quad \text{where } p_j = q_k = i \tag{2}$$

Of course, other statistical association measures based on rankings and not on actual parameter values may be used to compute the distance between two permutations. The two best-known examples of these are *Spearman's rho* (Siegel and Castellan, 1988), (which is equal to the deviation distance except that the deviations are squared and the result is expressed as a number between -1 and 1) and *Kendall's tau* (Kendall and Gibbons, 1990) (equal to the number of pairwise adjacent swaps needed to transform

the first permutation into the second one). In the metaheuristics literature, Martí et al. (2005) develop the R-type distance, usable when the *relative* ordering of the items in the solutions is more important. An example is an encoding for the TSP in which p_i represents the index of the city that is visited immediately after city i. This distance measure is defined as the number of times p_{i+1} does not immediately follow p_i in q, or formally

$$d_{\mathrm{R}}(p,q) = \sum_{i=1}^{n-1} y_i$$
$$\text{where} \quad y_i = \begin{cases} 1 & \text{if } \nexists j : p_i = q_j \text{ and } p_{i+1} = q_{j+1} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Many other distance measures that can be used for permutation problems exist, some of them are very difficult to calculate. The *reversal distance* for example, is defined as the number of substring reversals required to transform a string into another one. This distance measure is very important in molecular biology, but calculating it is $\mathcal{NP}$ hard (Caprara, 1997).

Although a wealth of distance measures is available that can calculate the distance between two permutations, this does not address the fact that solutions of a large number of problems cannot be represented as a simple permutation of items. Consider e.g. the classical capacitated vehicle routing problem (CVRP): not only does a solution to this problem consist of multiple permutations (the different routes), but these routes can be executed forwards and backwards. This in turn causes a new problem: a solution to a vehicle routing problem can be encoded in many different ways, e.g. by changing the order of the tours in the solution or by reversing some tours. A distance measure for the VRP should be able to take this into account and recognize the fact that the distance between two solutions encoded in different ways should be zero. In this paper, we use a single distance measure and modify it so it becomes usable for a large number of permutation problems. We define three types of extensions of the unmodified edit distance measure, that can be combined to be used for more complex solution representations. These three extensions are the following:

1. representations in which the reversed string represents the same solution (*reversal independence*),
2. representation in which a cycling of the string represents the same solution (*cyclic independence*) and
3. representations with *independent parts*.

The distance measure used throughout this paper is the *edit distance*, also called *Levenshtein distance*. This distance measure is based on the idea that the distance between two solutions is equal to the "cost" required to transform the first solution into the second one. We show that for a large number of combinatorial problems, an effective distance measure can be developed by adapting the edit distance measure to the specific requirements of the problem representation. The usefulness of some of the distance measures discussed in this paper is demonstrated by Janssens (2004),

who shows how they can be used in various aspects related to the design of metaheuristics for scheduling problems, without discussing their calculation. To the best of our knowledge, the only extension of which the calculation has been considered in the literature is the extension with cyclic independence, introduced by Maes (1990), who also develops an efficient algorithm for its calculation. In this paper, we provide formulas for the calculation of the distance between solutions of several scheduling and routing problems and prove some lemmas to simplify the calculation of these distance measures.

1 Edit distance

The edit distance is based on—and sometimes used as a synonym for—the *Levenshtein distance*, which was first introduced by Levenshtein (1966) in the context of error-correcting codes. When binary strings are transmitted across a channel, three types of errors can occur (Λ denotes the absence of a character).

- *Reversals*: errors of the type $1 \rightarrow 0$ or $0 \rightarrow 1$
- *Deletions*: errors of the type $1 \rightarrow \Lambda$ or $0 \rightarrow \Lambda$
- *Insertions*: errors of the type $\Lambda \rightarrow 0$ or $\Lambda \rightarrow 1$

Definition 1. The edit distance between two binary strings b_1 and b_2 is the minimal number of edit operations (reversals, insertions, deletions) required to transform string b_1 into string b_2.

Wagner and Fischer (1974) extend the work of Levenshtein (1966) in several ways. Strings are allowed to be composed of any finite alphabet (and not just a binary one). Levenshtein's *reverse* operation now becomes a *substitution* operation that occurs when a character is changed into another one. Secondly, different weights can be assigned to each individual edit operation. This turns Levenshtein's distance measure into a real-valued function of the two strings involved. Finally, Wagner and Fischer (1974) provide a dynamic programming algorithm to calculate their distance measure in time proportional to the product of the lengths of the strings. Under some mild assumptions about the weights assigned to each edit operation, the edit distance can be shown to be a metric.

1.1 Definitions

In the rest of this paper, we adhere to the following definitions. We assume that we are given a set S of *strings* composed of *characters* of a finite *alphabet* A. The length of a string s is written as $|s|$. The i-th character of string s is written as $s(i)$.

Λ is the *null-character*, signifying the absence of a character, i.e. $|\Lambda| = 0$.

Definition. An *elementary edit operation* $(x, y) \neq (\Lambda, \Lambda)$ is an ordered pair of characters from the set $A \cup \Lambda$. An elementary edit operation is called a *substitution* iff $x \neq \Lambda$ and $y \neq \Lambda$. It is called an *insertion* iff $x = \Lambda$ and a *deletion* iff $y = \Lambda$.

Definition. A sequence of edit operations that may be used to transform a string s into a string t is called an *edit transformation* of s into t.

Note that the concept of elementary edit operation does not include any information on where in the string the operation is supposed to take place. An edit transformation of s into t therefore does not necessarily specify unambiguously how exactly it should be used to transforms s into t, as there may be several possible ways to achieve this. Also, an edit transformation of s into t may contain redundant operations, e.g. a character that is inserted and later removed.

Elementary edit operations can be weighted by a weight function γ, that assigns a nonnegative real number $\gamma(x, y)$ to each elementary edit operation (x, y). Using the function γ, a nonnegative real value can be assigned to any edit transformation $E = [(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)]$. The weight of edit transformation E is equal to $\gamma(E) = \sum_{i=1}^{m} \gamma(x_i, y_i)$.

Definition. The *edit distance* $\delta(s, t)$ between strings s and t is defined as

$$\delta(s, t) = \min\{\gamma(E) | E \text{ is an edit transformation of } s \text{ into } t\}. \tag{4}$$

If $\gamma(x, y) = 1$ for all x, y then $\delta(s, t)$ is equal to the number of edits required to transform string s into t. Without loss of generality, the examples in the paper use a weighting function $\gamma(x, y) \equiv 1$.

Wagner and Fischer (1974) show that the concept of edit distance can be related to a structure called *trace*. Informally, a trace is a description of how an edit transformation transforms a string s into a string t, ignoring the order of edit operations and any possible redundancies. A trace $T_{s,t}$ is a sequence of pairs (i, j) where i is a position in string s and j is a position in string t. A trace can therefore be visualized as a set of lines connecting characters from s to characters from t that are either substituted (when they are different) or left unchanged (when they are equal). Characters in s and t that are not connected through a line in $T_{s,t}$ are deleted or added respectively. An example of a trace and its corresponding edit transformation is given in Fig. 1. In this figure, we also show how the edit transformation may be used to transform s into t. For a formal definition of a trace, we refer to Wagner and Fischer (1974).

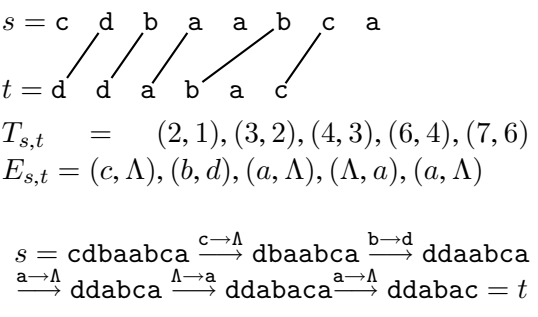

Fig. 1 Trace and edit transformation of $s = \texttt{cdbaabca}$ to $t = \texttt{ddabac}$

Let T be a trace from s to t. Let I and J be the sets of positions in s and t respectively not connected by a line in T, then the cost of trace T can be defined as

$$\text{cost}(T) = \sum_{(i,j)\in T} \gamma(s(i), t(j)) + \sum_{i\in I} \gamma(s(i), \Lambda) + \sum_{j\in J} \gamma(\Lambda, t(j)) \tag{5}$$

In other words, the cost of a trace from s to t is the cost of the edit transformation from s to t which consists of a substitution for each pair (i, j) in the trace, an insertion for each element of t not connected by a line in T and a deletion for each element in s not connected by a line in T.

Wagner and Fischer (1974) relate traces to edit transformations by showing that the edit distance $\delta(s, t)$ is equal to the minimum-cost trace from s to t.

$$\delta(s, t) = \min\{\text{cost}(T) | T \text{ is a trace from } s \text{ to } t\}. \tag{6}$$

The concept of a trace will prove useful later on, when proving that the calculations of some of the edit distance extensions can be simplified.

1.2 Algorithms and complexity

For two strings s and t of length $|s|$ respectively $|t|$, the worst-case time-complexity of the dynamic programming algorithm proposed by Wagner and Fischer (1974) is $O(|s|.|t|)$, i.e. $O(n^2)$ if the lengths of both strings is n. The *space* complexity can be reduced to $O(n)$ if only the value of the edit distance is needed (and not the edit sequence that this distance corresponds to).

A more efficient algorithm is given by Ukkonen (1985). The worst case time complexity for this algorithm is $O(nd)$, the average complexity is $O(n + 2d)$, where n is the length of the strings, and d is their edit distance. This is fast for similar strings where d is small, i.e. when $d \ll n$.

Other, even more efficient algorithms have been proposed, but for the purposes of this paper (i.e. the algorithms are used to determine the distance between solutions of combinatorial optimization problems, that are of relatively small size) these algorithms are unnecessarily intricate.

2 Application of the edit distance to permutation-type representations

For problems of which a solution can be represented as a simple permutation (with or without repetition) of the elements of some set of problem attributes, the edit distance in its original form can be used. Permutations with repetition are appropriate when a certain item can appear several times in the solution. Bierwirth (1995), e.g. studies a job shop problem on several machines and uses a permutation with repetitions to accomodate the fact that some jobs need to be processed on more than one machine and may occur several times in a solution.

Example. A single machine scheduling problem requires the scheduling of 8 jobs labeled `1` to `8`. The distance between a solution $s =$ `52643187` and a solution

$t = 65321487$ can be calculated as $\delta(s, t) = \delta(52643187, 65321487) = 5$. To transform s into t in 5 edit operations, remove 3 and 1 from s, change 6 into 1 and then add 6 and 3 in the appropriate locations.

For more complex representations however, the edit distance cannot be used in its standard form. The main reason for this is that in some problems the permutational representation of a solution is not unique. In other words: several representations correspond to the same solution and the distance between two such representations of the same solution should be zero.

2.1 Representations with reversal independence

Some solution representations exhibit *reversal independence*, meaning that the reverse string represents the same solution.

Consider a path problem the object of which is to find the edges that connect two nodes n_{first} and n_{final}. A solution can be represented by an ordered set of edges $e_1 e_2 \ldots e_m$. The distance between two solutions can be determined by using the edit distance (using the edge set as alphabet). However, a path using edges $e_1 e_2 \ldots e_m$ in this order and running from edge e_1 to edge e_m is equivalent to the reverse path, represented as $e_m \ldots e_2 e_1$. Any distance measure comparing these two paths should return zero.[1]

Example. Given the graph in Fig. 2, two paths between n_{first} and n_{final} can be found. The first one is $p_1 = \texttt{abd}$, the second one is $p_2 = \texttt{cd}$.

It is clear that `abd` and `dba` are both valid representations of p_1. Also, `cd` and `dc` are both representations of p_2. The distance between p_1 and p_2 should reflect this. Figure 3 shows the possible edit distances (assuming all weights are equal to one).

The "correct" edit distance is 2, the smallest possible distance in Fig. 3. Under this interpretation, it holds that $\delta(\texttt{abd}, \texttt{cd}) = \delta(\texttt{dba}, \texttt{dc})$ and $\delta(\texttt{abd}, \texttt{dc}) = \delta(\texttt{dba}, \texttt{cd})$. This is a general property that simplifies the calculation of the edit distance in the case of reversal independence.

Fig. 2 A shortest path problem

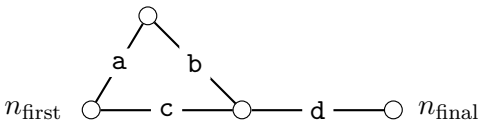

Fig. 3 Possible edit distances for the shortest path problem

	abd	dba
cd	2	3
dc	3	2

[1] Admittedly all paths have a "natural" direction and all representations can be oriented in the same direction (from the first node to the final node). However, this is not the case in all problems (e.g. routing problems).

Edit distance for representations with reversal independence

Let the reverse string of a string s be given by $\tilde{s}$, then the modified edit distance for representations with reversal independence is

$$\begin{aligned}\delta_{\text{ri}}(s,t) &= \min(\delta(s,t), \delta(s,\tilde{t})) \\ &\equiv \min(\delta(s,t), \delta(\tilde{s},t)). \end{aligned} \tag{7}$$

Like the original edit distance measure, this measure can be calculated in $O(n^2)$. The equivalence sign in Eq. (7) signifies that the edit distance between a string s and the reverse of a second string t is equal to the edit distance between the reverse of s and t. This is proved using the following lemmas.

Lemma 1. *Any edit transformation that can be used to transform a string s into a string t can also be used to transform $\tilde{s}$ into $\tilde{t}$.*

Proof: This follows from the definitions of edit transformation and elementary edit operation. □

Lemma 2. *The edit distance between two strings does not change when both strings are reversed, i.e.*

$$\delta(s,t) \equiv \delta(\tilde{s},\tilde{t}).$$

Proof: From Lemma 1, it follows that $\delta(\tilde{s},\tilde{t}) \leq \delta(s,t)$ (as exactly the same elementary edit operations can be used to transform $\tilde{s}$ into $\tilde{t}$, the edit distance between these two strings cannot be greater than the edit distance between s and t). Suppose now that $\delta(\tilde{s},\tilde{t}) < \delta(s,t)$. According to Lemma 1, this would imply that s can be transformed into t in less edits that its edit distance, which is a contradiction. As a result, $\delta(\tilde{s},\tilde{t}) = \delta(s,t)$. □

Lemma 3. *The edit distance between a string and the reverse of another string is equal to the edit distance between the reverse of the first string and the second string, formally*

$$\delta(s,\tilde{t}) \equiv \delta(\tilde{s},t). \tag{8}$$

Proof: This result follows from Lemma 2 by setting t equal to $\tilde{t}$ and remarking that $\tilde{\tilde{s}} = s$. □

This result simplifies the calculation of the edit distance in the case of reversal independence. It is not necessary to compare each string with the reverse of the other, it suffices to compare just one string with the reverse of the other.

2.2 Representations with cyclic independence

Some solution representations exhibit the property of *cyclic independence*. For these representations, a solution can be encoded as a permutation, but only the relative position is important. An example of a problem for which such a representation occurs naturally is the *traveling salesman problem on a directed graph*, for which solutions are encoded as a string of customers. For this problem, two strings that have the same order of customers but start at a different position, represent the same solution and should consequently have an edit distance of zero. E.g. for a traveling salesman problem with customers c_1 to c_m, strings $c_1c_2 \ldots c_ic_j \ldots c_m$ and $c_j \ldots c_mc_1c_2 \ldots c_i$ represent the same tour. If the traveling salesman problem is defined on an undirected graph, the representation exhibits reversal independence in addition. A commonly used distance measure for the traveling salesman problem is the number of common edges in two solutions. This measure has the disadvantage that it does not explicitly take into account the order in which the customers appear in the tour. In some situations, the edit distance with cyclic independence might therefore be preferable.

Example. Given a traveling salesman problem with 5 customers labeled `a` to `e`. Two different tours are found, represented respectively as $t_1 =$ `abcde` and $t_2 =$ `adbce`. The possible edit distances (again assuming all weights are equal to 1). Are found in Fig. 4.

From this figure, it follows that the minimal number of edits required to transform a cycling of t_1 into one of t_2 is 2. Note that this minimal distance can be found at least once in each column and each row. This is a general property, that will be proved below and that allows us to simplify considerably the calculation of the edit distance in the case of cyclic independence. This example also proves that it is not sufficient to apply a convention to avoid cyclic independence, such as choosing the cycling of solutions that start with the same customer. In our example, choosing `a` as the first customer would result in a distance of 2, whereas choosing `d` as the first customer would give a distance of 4.

Edit distance for representations with cyclic independence

Assume that $|s| = n$ and let $s(i)$ be the i-th character of string s. s^i represents a cycling, starting from position i, i.e. $s^i = s(i)s(i+1)\ldots s(n)s(1)s(2)\ldots s(i-1)$. The

	`abcde`	`bcdea`	`cdeab`	`deabc`	`eabcd`
`adbce`	2	4	4	4	3
`dbcea`	3	2	4	4	4
`bcead`	4	2	3	4	4
`ceadb`	4	4	2	3	4
`eadbc`	4	4	4	2	2

Fig. 4 Possible edit distances for the traveling salesman problem

modified edit distance for representations with cyclic independence is

$$\delta_{\mathrm{ci}}(s,t) = \min_i \delta(s^i, t) \equiv \min_j \delta(s, t^j). \tag{9}$$

If both strings are approximately of size n, this measure can be calculated by enumeration in $O(n^3)$.

The equivalence sign in Eq. (9) indicates that the minimum distance between s and all cyclings of t is equal to the minimum distance between t and all cyclings of s. This can be proved using the following lemmas.

Lemma 4. *If the minimum distance between strings s and t that have cyclic independence is given by $\delta(s,t) = \delta(s^i, t^j)$ (i.e. no cycling exists that has a smaller edit distance), then a cycling t^p of t exists for which $\delta(s^{i+1}, t^p) = \delta(s,t)$*

Proof: Let $|s| = n$ and $|t| = m$. Let T_{s^i,t^j} be a minimum-cost trace between s^i and t^j. Two possibilities arise: (1) $s(i)$ is not connected by T_{s^i,t^j} to a character of t^j and (2) $s(i)$ is connected by T_{s^i,t^j} to a character of t^j.

In the first case, the edit transformation corresponding to the trace T_{s^i,t^j} will result in the removal of $s(i)$ from s^i. The edit distance between s^{i+1} and t^j therefore cannot be larger than that between s^i and t^j because removing $s(i)$ from s^{i+1} yields the same string as removing $s(i)$ from s^i and the trace remains the same. The edit distance between s^{i+1} and t^j cannot be smaller because that would violate our assumption that $\delta(s^i, t^j)$ is minimal. In this case, setting $p = j$ proves the lemma.

In the second case, assume that $s(i)$ is connected by the trace to $t(q)$ (see Fig. 5). The edit transformation of s^i to t^j will delete $t(j)$ to $t(q-1)$ and change $s(i)$ into $t(q)$ (if they represent different characters). To transform s^{i+1} into t^{q+1}, the same operations can be used. Removing $t(j)$ to $t(q-1)$ from both t^j and t^{q+1} yields the same string. The edit distance between s^{i+1} and t^{q+1} therefore cannot be larger than $\delta(s,t)$. It cannot be smaller, because this would violate the assumption that $\delta(s^i, t^j)$ is minimal. Therefore $\delta(s^{i+1}, t^{q+1}) = \delta(s,t)$. In this case, setting $p = q+1$ proves the lemma. Fig. 5 shows this. □

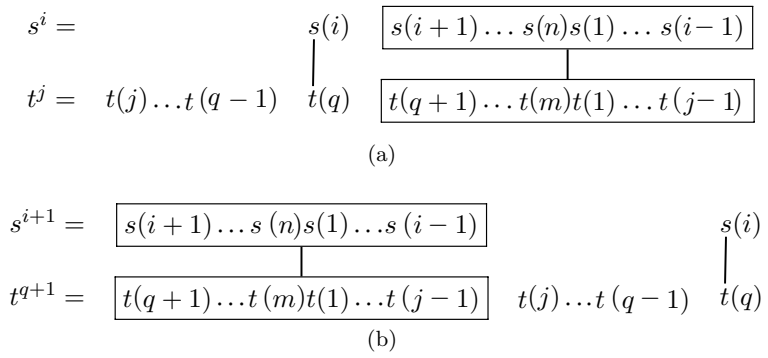

Fig. 5 Proof of Lemma 4—Case 2

Lemma 5. *If the minimum distance between strings s and t that have cyclic independence is given by $\delta(s,t) = \delta(s^i, t^j)$ (i.e. no cycling exists that has a smaller edit distance), then for all r $(1 \leq r \leq |s|)$ a cycling t^p of t exists for which $\delta(s^r, t^p) = \delta(s,t)$*

Proof: The proof of this lemma follows from the repeated application of Lemma 4. □

A consequence of Lemma 5 is that the edit distance between two strings that exhibit cyclic independence can be calculated by comparing any cycling of a string to all cyclings of the other string.

2.3 Representations with independent parts

Some problem representations consist of parts that are independent. Depending on the specific representation, different modifications of the edit distance are possible. The extensions mentioned here are not specific for the edit distance and may be used with any permutation distance measure.

2.3.1 Representations with recognizable parts

In some cases, the parts of the solution represent different recognizable entities. Stated differently, each part of s has a "naturally" corresponding part in t. Examples can be found in multiple machine scheduling, in which each solution has a sequence of jobs scheduled on each machine. A natural modification of the edit distance measure for this representation is to measure the edit distance between the corresponding parts in both solutions and add the distances.

Suppose solution s is split into parts σ_1 to σ_n and t is split into parts τ_1 to τ_n, then

$$\delta_{\text{rp}}(s,t) = \sum_{i=1}^{n} \delta(\sigma_i, \tau_i). \tag{10}$$

2.3.2 Representations with unrecognizable parts

If the parts of the solution are not recognizably linked to a certain entity, it is impossible to determine in advance which part of solution 1 should correspond to which part of solution 2. The "ideal" correspondence between parts of s and parts of t lies in the fact that this correspondence minimizes the total distance. This involves solving an assignment problem. Suppose solution s is split into parts $\sigma_1, \sigma_2, \ldots, \sigma_m$ and t is split into parts $\tau_1, \tau_2, \ldots, \tau_n$. The modified edit distance measure is then given by the following mathematical program.

$$\delta_{\text{up}}(s,t) = \min \sum_{i=1}^{n} \sum_{j=1}^{m} \delta(\sigma_i, \tau_j) x_{ij}, \tag{11a}$$

s.t.

$$x_{ij} \in 0, 1 \quad \forall i, j, \tag{11b}$$

$$\sum_j x_{ij} = 1 \quad \forall i, \tag{11c}$$

$$\sum_i x_{ij} = 1 \quad \forall j. \tag{11d}$$

where x_{ij} is 1 if part i of solution 1 is assigned to part j of solution 2 and 0 otherwise.

Example. In the vehicle routing problem with time windows, a central depot is given, as well as a set of customers requiring service. The object of this problem is to build routes that start and end at a depot and optimize one or more performance measures (number of vehicles used, total travel time, etc.). Moreover, each customer has a certain time window, that determines the earliest and latest time the service at this customer can start.

A solution to a vehicle routing problem with time windows can be represented as a set of sequences that have cyclic independence, but no reversal independence (i.e. the tours cannot be reversed). Each sequence represents a tour, starting and ending at the depot.

For a vehicle routing problem with 10 customers, labeled `a` to `j`, 2 solutions are given. $s =$ (`abc`)(`defg`)(`hij`) and $t =$ (`bcdef`)(`gjiha`).

Solving an assignment problem with the matrix in Fig. 6 as cost matrix assigns `abc` $\rightarrow \Lambda$, `defg` $\rightarrow$ `bcdef` and `hij` $\rightarrow$ `gjiha` (indicated in bold), yielding a total cost of 10.

3 Conclusions

In this paper, we have developed several extensions of the edit distance measure, that can be used to measure the similarity or difference between solutions of combinatorial problems, that can be represented as (sets of) permutations. We have shown how each of these extensions can be calculated and have proved some lemmas to simplify their calculation.

It should be mentioned that some problem representations require the combination of several extensions. The vehicle routing problem on an undirected network, e.g., exhibits reversal independence as well as unrecognizable independent parts. The edit distance between a pair of VRP solutions can be found by calculating the reversal independent edit distance between each pair consisting of a trip from the first solution

Fig. 6 Possible edit distances for the vehicle routing problem with time windows

	`abc`	`defg`	`hij`
`bcdef`	4	**3**	5
`gjiha`	5	5	**4**
Λ	**3**	4	3

and a trip from the second solution and using these values to calculate the edit distance for independent, unrecognizable parts.

Future research might focus on using these modified edit distances in the design of evolutionary and other algorithms. Another interesting topic for future research is to similarly extend other—perhaps less computationally demanding—distance measures.

References

Bierwirth, C. (1995). "A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms." *OR Spektrum* 17, 87–92.

Caprara, A. (1977). "Sorting by Reversal is Difficult." In *Proceedings of the the First Annual International Conference on Computational Molecular Biology (RECOMB'97)*, ACM Press, New York, pp. 75–83.

Glover, F., M. Laguna, and R. Martí. (2000a). "Fundamentals of Scatter Search and Path Relinking." *Control and Cybernetics* 39, 653–684.

Glover, F., A. Løkketangen, and D.L. Woodruff. (2000b). "Scatter Search to Generate Diverse MIP Solutions." In M. Laguna and J.L.G. Velarde (eds.), *Computing Tools for Modeling Optimization and Simulation*, Kluwer, Boston, pp. 299–320.

Goldberg, D.E. and J. Richardson. (1987). "Genetic Algorithms with Sharing for Multimodal Function Optimization." In *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Inc, Mahwah, NJ, pp. 41–49.

P. Greistorfer and S. Voß. (2005). "Controlled Pool Maintenance for Metaheuristics." In C. Rego and B. Alidaee (eds.), *Metaheuristic Optimization Via Memory and Evolution*, Kluwer, Boston, pp. 387–424.

Janssens, G.K. (2004). "A Proposition for a Distance Measure in Neighbourhood Search for Scheduling Problems." *Journal of the Chinese Institute of Industrial Engineers* 21, 262–271.

Kendall, M. and J. Dickinson Gibbons. (1990). *Rank Correlation Methods*. Oxford University Press, New York.

Levenshtein, V.I. (1966). "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals." *Soviet Physics-Doklady* 10, 707–710.

Maes, M. (1990). "On a Cyclic String-to-String Correction Problem." *Information Processing Letters* 35, 73–78.

Mahfoud, S.W. (1992). "Crowding and Preselection Revisited." In R. Manner and B. Manderick (eds.), *Parallel Problem Solving from Nature*. Elsevier, Amsterdam, pp. 27–36.

Martí, R., M. Laguna, and V. Campos. (2005). "Scatter Search vs. Genetic Algorithms: An Experimental Evaluation with Permutation Problems." In C. Rego and B. Alidaee (eds.), *Metaheuristic Optimization Via Adaptive Memory and Evolution: Tabu Search and Scatter Search*, Kluwer Academic Publishers, Boston, pp. 263–282.

Mauldin, M. (1984). "Maintaining Diversity in Genetic Search." In *Proceedings of the National Conference on Artificial Intelligence*, pp. 247–250.

Ronald, S. (1998). "More Distance Functions for Order-Based Encodings." In *Proceedings of the IEEE Conference on Evolutionary Computation*. IEEE Press, New York, pp. 558–563.

Siegel, S. and N.J. Castellan. (1988). *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill, London.

Sörensen, K. and M. Sevaux. (2006). "MA|PM: Memetic Algorithms with Population Management." *Computers and Operations Research* 33, 1214–1225.

Ukkonen, E. (1985). "Finding Approximate Patterns in Strings." *Journal of Algorithms* 6, 132–137.

Wagner, R.A. and M.J. Fischer. (1974). "The String-to-String Correction Problem." *Journal of the Association for Computing Machinery* 21, 168–173.