ARTICLE IN PRESS

Discrete Applied Mathematics ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Computing similarity distances between rankings☆

Farzad Farnoud (Hassanzadeh)[a,b], Olgica Milenkovic[c], Gregory J. Puleo[d,*], Lili Su[c]

[a] Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA, USA
[b] Department of Computer Science, University of Virginia, Charlottesville, VA, USA
[c] Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA
[d] Coordinated Science Lab, University of Illinois at Urbana-Champaign, Urbana, IL, USA

ARTICLE INFO

Article history:
Received 19 April 2016
Received in revised form 25 July 2017
Accepted 28 July 2017
Available online xxxx

Keywords:
Permutations
Similarity distance
Transposition distance

ABSTRACT

We address the problem of computing distances between permutations that take into account similarities between elements of the ground set dictated by a graph. The problem may be summarized as follows: Given two permutations and a positive cost function on transpositions that depends on the similarity of the elements involved, find a smallest cost sequence of transpositions that converts one permutation into another. Our focus is on costs that may be described via special metric-tree structures. The presented results include a linear-time algorithm for finding a minimum cost decomposition for simple cycles and a linear-time 4/3-approximation algorithm for permutations that contain multiple cycles. The proposed methods rely on investigating a newly introduced balancing property of cycles embedded in trees, cycle-merging methods, and shortest path optimization techniques.

1. Introduction

The problem of sorting distinct elements according to a given set of criteria has a long history and has been studied in mathematics, computer science, and social choice theory alike [8,11,19]. One volume of the classical text in computer science – Knuth's *The Art of Computer Programming* – is almost entirely devoted to the study of sorting. The solution to the problem is well known when the sorting steps are swaps (transpositions) of two elements: In this case, it is convenient to first perform a cycle decomposition of the permutation and then swap elements in the same cycle until all cycles have unit length.

Sorting problems naturally introduce the need for studying *distances* between permutations. There are many different forms of distance functions on permutations, with the two most frequently used being the Cayley distance and the Kendall distance [5]. Although many generalizations of the Cayley, Kendall and other distances are known [16], only a handful of results pertain to distances in which one assigns positive weights or random costs[1] to the basic rearrangement steps [1,6,14]. Most such work has been performed in connection with genome rearrangement studies [2,7] and for the purpose of gene prioritization [18]. (Note that [2,7] use a different notion of "transposition" than is used in this paper.) Some other examples appear in the social sciences literature (see references in [6]), pertaining to constrained vote aggregation and logistics [12].

☆ This work was supported in part by the NSF STC Class 2010 CCF 0939370 grant. Research of the third author is supported by the IC Postdoctoral Research Fellowship (grant number 2014-14081100005). Farzad Farnoud was with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA.

* Corresponding author.
E-mail address: gjp0007@auburn.edu (G.J. Puleo).

[1] Throughout the paper, we use the words cost and weight interchangeably, depending on the context of the exposition.

http://dx.doi.org/10.1016/j.dam.2017.07.038
0166-218X/

A related line of work is the study of sorting algorithms in which *comparisons* between different objects may have different costs depending on the objects in question. Such problems were studied (in a broader context) by Charikar et al. in [3,4], and this line of work was continued by [9,13] who considered additional constraints on the cost functions involved. Gupta and Kumar [10] considered the problem of sorting objects in the case where the comparison costs form a *metric* on the ground set. This is of particular relevance to our work in this paper, where we consider transposition costs which form a metric on the ground set. However, all of the problems in this line of work deal with nonuniform costs on *information*, while we instead impose nonuniform costs on the rearrangement steps, without imposing any costs on information. As such, while the broad ideas are similar, the technical details differ greatly between this line of research and the current work.

A number of practical problems call for positive costs (weights) on transpositions, and costs that capture some constraint on the structure of the transpositions. The problem at hand may then be described as follows: For a given set of positive costs assigned to transpositions of distinct elements, find a smallest cost sequence of transpositions converting a given permutation to the identity.

In our subsequent analysis, we focus on constraints that take into account that elements of the ground set may be similar and that transposing similar elements should induce a smaller cost than transposing dissimilar elements. We refer to the underlying family of distance measures as *similarity distances*. The similarity distance is not to be confused with the distance used in [17], where the goal was to *rank similar items* close to each other in an aggregated list.

The contributions of this work are three-fold. First, we introduce a Y-tree (i.e., a tree with at most one node of degree three) cost function and a notion of similarity between permutations associated with this special tree structure. In this setting, the cost of transposing two elements equals the weight of the shortest path in a Y-tree. Our focus on Y-trees is largely motivated by the fact that the general tree analysis appears to be quite complex. While Y-trees are simple enough to be computationally tractable, they are complex enough that interesting new phenomena arise that are not present in path metrics. Second, we describe an exact linear time decomposition algorithm for cycle permutations with Y-tree costs. Third, we develop a linear time, 4/3-approximation method for computing the similarity distance between arbitrary permutations.

The paper is organized as follows. Section 2 introduces the notation and definitions used throughout the paper. Section 3 contains a brief review of prior work as well as some relevant results used in subsequent derivations. This section also presents a linear time algorithm for computing the Y-tree similarity between cycle permutations. This algorithm is extended in Section 4 to general permutations via cycle-merging strategies that provide linear time, constant-approximation guarantees. Section 5 contains the concluding remarks.

2. Mathematical preliminaries

For a given ground set $[n] \triangleq \{1, 2, \ldots, n\}$, a permutation $\pi : [n] \to [n]$ is a bijection on and onto $[n]$. The collection of all permutations on $[n]$ – the symmetric group of order $n!$ – is denoted by S_n.

There are several ways to represent a permutation. The two-line representation has the domain written out in the first line and the corresponding image in the second line. For example, the following permutation is given in two-line form:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 1 & 2 & 5 & 4 & 3 \end{pmatrix}.$$

The one-line representation is more succinct as it only utilizes the second row of the two-line representation; the above permutation in one-line format reads as (6, 1, 2, 5, 4, 3). The symbol e is reserved for the identity permutation $(1, 2, \ldots, n)$.

Sometimes, we find it useful to describe a permutation in terms of elements and their images: hence, a third description of the aforementioned permutation is $\pi(1) = 6$, $\pi(2) = 1$, $\pi(3) = 2$, $\pi(4) = 5$, $\pi(5) = 4$, and $\pi(6) = 3$. A straightforward interpretation of these expressions is that $\pi(i)$ represents the element placed in position i. We also define the inverse of a permutation π, π^{-1}, in which $\pi^{-1}(i)$ describes the position of element i. With this notation at hand, the product of two permutations $\pi, \sigma \in S_n$, $\mu = \pi\,\sigma$, can be defined by $\mu(i) = \pi(\sigma(i))$, for all $i \in [n]$. The *support* of a permutation $\pi \in S_n$, written $\mathrm{supp}(\pi)$, is the set of all $i \in [n]$ with $\pi(i) \neq i$. We write $|\pi|$ to refer to $|\mathrm{supp}(\pi)|$.

For $k > 1$, a *k-cycle*, denoted by $\kappa = (i_1 \ \ldots \ i_k)$, is a permutation that acts on $[n]$ in the following way[2] :

$$i_1 \to i_2 \to \ldots \to i_k \to i_1,$$

where $x \to y$ denotes $y = \kappa(x)$. In other words, $\kappa = (i_1 \ \ldots \ i_k)$ cyclically shifts elements in the permutation confined to the set $\{i_1, \ldots, i_k\}$ and keeps all other elements fixed. A cycle of length 2 is called a *transposition*, and is denoted by $(a\,b)$.

In general, for $a, b \in [n]$, $\pi(a\,b) \neq (a\,b)\pi$, because $\pi(a\,b)$ corresponds to swapping elements of π in positions a and b while $(a\,b)\pi$ corresponds to swapping elements a and b in π. For instance, $(6, 1, 2, 5, 4, 3)(2\,3) = (6, 2, 1, 5, 4, 3)$, while $(2\,3)(6, 1, 2, 5, 4, 3) = (6, 1, 3, 5, 4, 2)$. Note that in the former example, we used $\pi(a\,b)$ to denote the product of a permutation and a transposition.

Two cycles are said to be *disjoint* if the intersection of their supports is empty; furthermore, two cycles are termed to be *adjacent* if they have exactly one common element in their supports. Although non-disjoint cycles are sporadically mentioned in the combinatorial literature, their use is extremely limited due to the fact that disjoint cycles offer simpler

[2] This is not to be confused with the one line representation using commas between entries.

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.

means to study problems on permutations. In particular, the concept of adjacent cycles was, to the best of the authors' knowledge, not previously used for analyzing sorting algorithms.

A permutation can be uniquely decomposed into a product of disjoint cycles, often referred to as the *cycle decomposition* or the *cycle representation*. For example, the cycle decomposition of the permutation (6, 1, 2, 5, 4, 3) equals (1 6 3 2)(4 5), where one can freely choose the order in which to multiply (1 6 3 2) and (4 5). More generally, we may wish to write a permutation π as a product $\pi_1 \cdots \pi_k$ of cycles so that for each $i < k$, the cycles π_i and π_{i+1} are either disjoint or adjacent. We call such a product an *adjacent cycle decomposition* of π. One significant difference between these two types of cycle decomposition is that in an adjacent cycle decomposition, the order of multiplication matters (i.e., the product is non-commutative); (1 6 3 2) equals (2 1 6)(3 6), but not (3 6)(2 1 6). As opposed to the disjoint cycle decomposition, which is unique up to the order of the disjoint cycles, there may exist multiple adjacent cycle decompositions of a given permutation.

The *functional digraph* of a function $f : [n] \to [n]$, denoted by $\mathcal{G}(f)$, is a directed graph with vertex set $[n]$ and arcs from i to $f(i)$ for each $i \in [n]$. Arcs are subsequently denoted by $(i \to f(i))$. For a permutation π, $\mathcal{G}(\pi)$ is a collection of disjoint cycles; hence, the cycles of the permutation correspond to the cycles of its functional digraph.

Given any connected, undirected, edge-weighted graph G on the vertex set $[n]$ with positive edge weights, we can define a metric φ by letting $\varphi(a, b)$ be the minimum weight of an a, b-path in G. If φ can be defined from G in this way, we say that φ is a *graph metric* and that G is a *defining graph* for φ. While in general the defining graph is unrestricted, we will typically be interested in graph metrics with a defining graph that falls into some special graph class. When φ is a metric on $[n]$, we also consider φ as giving weights to the transpositions in S_n, where the transposition $(a\, b)$ has weight $\varphi(a, b)$.

The weight of an ordered sequence of transpositions is defined as the sum of the weights of its constituent elements. That is, the weight of the sequence of transpositions $T = (\tau_1, \ldots, \tau_{|T|})$ equals

$$\mathrm{wt}_\varphi(T) = \sum_{i=1}^{|T|} \varphi(\tau_i) = \sum_{i=1}^{|T|} \varphi(a_i, b_i),$$

where τ_i denotes the transposition $(a_i\, b_i)$, and $|T|$ denotes the number of transpositions in the sequence T. When φ is understood (as will typically be the case throughout this paper) we suppress the subscripts and simply write $\mathrm{wt}(T)$. The same convention is used for all other notation involving the subscript φ.

If $\sigma = \pi\tau_1\tau_2 \ldots \tau_{|T|}$, we refer to $T = (\tau_1, \ldots, \tau_{|T|})$ as a *transform*, converting π into σ. The set of all such transforms is denoted by $A(\pi, \sigma)$. Clearly, $A(\pi, \sigma)$ is non-empty for any $\pi, \sigma \in S_n$. A transform that converts π into e, the identity permutation, is a *sorting* of π. On the other hand, a *decomposition* of π is a sequence $T = (\omega_1, \ldots, \omega_{|T|})$ of transpositions such that $\pi = \omega_1\omega_2 \ldots \omega_{|T|}$. Note that the minimum weight of a decomposition is the same as the minimum weight of a sorting as one sequence is equal to the other in reverse order.

The φ-weighted transposition distance between π and σ is defined by

$$d_\varphi(\pi, \sigma) = \min_{T \in A(\pi,\sigma)} \mathrm{wt}_\varphi(T).$$

Computing $d(\pi, \sigma)$ may be cast as a minimization problem over $A(\pi, \sigma)$, namely the problem of finding a *minimum cost transform* $T^* \in A(\pi, \sigma)$ such that $d(\pi, \sigma) = \mathrm{wt}(T^*)$. If $\varphi(a, b) = 1$ for all distinct a and b, the weighted transposition distance reduces to the well-known Cayley distance.

It is easy to verify that for every positive weight function, the weighted transposition distance d is a metric and furthermore, left-invariant (i.e., $d(\pi, \sigma) = d(\omega\pi, \omega\sigma)$). Hence, we may set one of the permutations (say, σ) to e, and write

$$\delta_\varphi(\pi) = d_\varphi(\pi, e) = \min_{T \in A(\pi,e)} \mathrm{wt}_\varphi(T).$$

We refer to the problem of computing $\delta(\pi)$ as the (weighted) *decomposition problem*.

With respect to the choice of weight functions, we restrict our attention to the previously introduced family of graph metric weights, satisfying the triangle inequality

$$\varphi(a, b) \le \varphi(a, c) + \varphi(c, b), \text{ for all distinct } a, b, c \in [n].$$

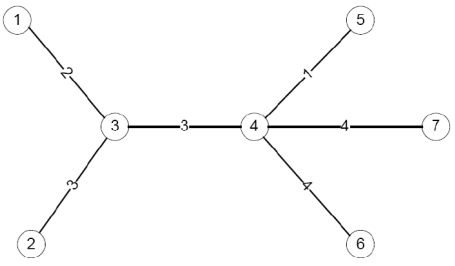
In particular, if we fix a tree-structured defining graph, the weight function φ is termed a *metric-tree* weight function. For such defining graphs, there clearly exists a unique minimum cost path between any two vertices, and for $a, b \in [n]$, $\varphi(a, b)$ is the sum of the weights of the edges on the unique path between a and b in G. If G is a path (line graph), then φ is called a *metric-path* weight function. If there exists a unique vertex in a tree-structured G of degree larger than or equal to three, the graph is called a *metric-star*. The vertex with highest degree is referred to as the *central vertex*. If the central vertex has degree three, the defining graph is called a *Y-tree*. The corresponding metric is referred to as the Y-tree metric. Examples of the aforementioned defining graphs are shown in Fig. 1.

The problem of finding $\delta(\pi)$ when φ is a metric-path weight was studied by the authors in [6]. The focus of the results to follow is on determining $\delta(\pi)$ when the defining graph is a Y-tree. The problem of evaluating $\delta(\pi)$ under a general metric-tree model appears difficult to handle by methods proposed in this work and will hence not be discussed.

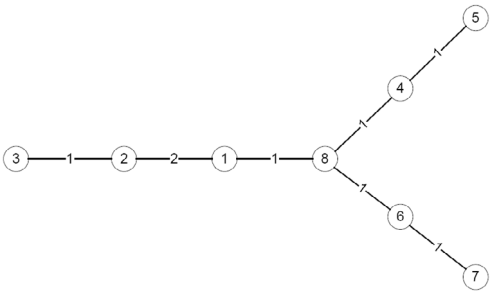
The following function, termed the *displacement*, is of crucial importance in our analysis of similarity distances on Y-trees:

$$D_\varphi(\pi, \sigma) = \sum_{i=1}^{n} \varphi(\pi^{-1}(i), \sigma^{-1}(i)).$$

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.



(1) A defining graph G corresponding to a general metric-tree weight function φ. The edge labels correspond to their underlying weights. From the graph, one reads $\varphi(1,7) = \varphi(1,3) + \varphi(3,4) + \varphi(4,7) = 2 + 3 + 4 = 9$.



(2) A defining graph G of a Y-tree with one edge cost equal to two, and all other edge costs equal to one. From the graph, one reads $\varphi(1,7) = \varphi(1,8) + \varphi(8,6) + \varphi(6,7) = 1 + 1 + 1 = 3$.

Fig. 1. Examples of defining graphs.

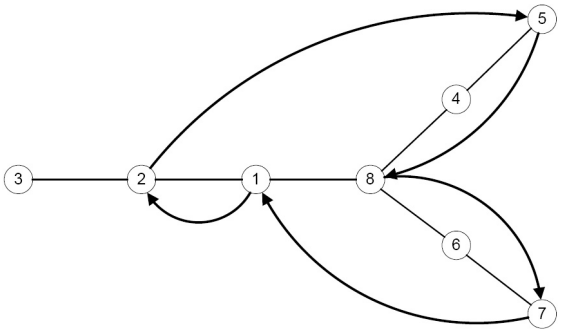


Fig. 2. Defining Y-tree and the cycle (1 2 5 8 7). Thin lines represent the defining Y-tree G, while boldfaced arcs represent the digraph of the cycle permutation, $\mathcal{G}$.

The displacement $D(\pi, \sigma)$ captures the overall cost of independently performing optimal transpositions of pairs of elements that are out of order. It is again easy to verify that for every positive weight function, the displacement $D(\pi, \sigma)$ is a metric and in addition, left-invariant (i.e., $D(\pi, \sigma) = D(\omega\pi, \omega\sigma)$, for all $\pi, \sigma, \omega \in S_n$). As a result, the notation and analysis may be simplified by assuming that $\sigma = e$ and by denoting the resulting displacement by $D(\pi)$.

The following properties of the displacement are easy to verify:

1. $D(\pi) = 0$ if and only if $\pi = e$.
2. $D(\pi_1\pi_2) \leq D(\pi_1) + D(\pi_2)$, for all permutations π_1 and π_2.
3. $D(\pi) = D(\pi^{-1})$, for all permutations π.

Consequently, we may write

$$D(\pi) = \sum_{i=1}^{n} \varphi(i, \pi(i)).$$

The main results of the paper are devoted to the study of decompositions of single cycles, for which we exactly determine the minimum cost of a decomposition. The focus on single cycles is justified by the approximation algorithm presented in Section 4, which shows that decomposing the individual cycles of a general permutation π yields a 4/3-approximation to an optimal decomposition of π.

For ease of exposition, we draw the digraph of a permutation and the undirected defining Y-tree graph of the given weight function on the same vertex set, as shown in Fig. 2. In this case, we say that the permutation is embedded in the defining graph. This graphical representation of both the cost function and the cycle decomposition of a permutation allows us to illustrate examples and gain intuition about the algorithms involved in the decomposition approach.

Denote the branches of a Y-tree, which are sets of nodes on paths starting from the central vertex and extending to a leaf, excluding the central vertex, by B_1, B_2, and B_3. Furthermore, for ease of exposition, denote the branch containing vertex v by $\mathrm{Br}(v)$. Next, we formalize the notion of a cycle lying on a path on the Y-tree as a cycle that has support contained in

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.

$B_i \bigcup B_j \bigcup \{v_c\}$, for some not necessarily distinct i, j, and with v_c representing the central vertex. In other words, a cycle lies on a path if its support is contained in the union of at most two of the three branches and the central vertex.

For a branch pair (B_i, B_j), $i \neq j$, let l_{ij}^{π} be the number of arcs from B_i to B_j in π; similarly, let l_{ji}^{π} be the number of arcs from B_j to B_i in π. If it is clear from the context, the superscript π will be omitted. A permutation is *balanced* if $l_{ij} = l_{ji}$ for all $i, j \in \{1, 2, 3\}$, and *unbalanced* otherwise.

For a permutation π and cost function φ, the *inefficiency* $\Delta_{\varphi}(a, b; \pi)$ of a transposition $(a\, b)$ with respect to π and φ is defined by

$$\Delta_{\varphi}(a, b; \pi) = 2\varphi(a, b) - (D(\pi) - D(\pi(a\, b))).$$

When $\tau = (a\, b)$, we also write $\Delta_{\varphi}(\tau; \pi)$ for $\Delta_{\varphi}(a, b; \pi)$.

The intuition behind the notion of inefficiency comes from the observation that a transposition $(a\, b)$ can reduce the overall displacement by at most $2\varphi(a, b)$; the inefficiency measures the gap from the optimal reduction. Also, since $2\varphi(a, b) = D((a, b))$, it follows that the inefficiency is nonnegative. Henceforth, a transposition $(a\, b)$ is termed *efficient* with respect to π if $\Delta(a, b; \pi) = 0$ and *inefficient* if $\Delta(a, b; \pi) > 0$.

The rest of the paper is organized as follows. In Section 3, we derive a closed form expression for a minimum cost of a decomposition of a single cycle and present an exact algorithm that can find the minimum cost decomposition T^* in linear time. In Section 4, we develop a linear time 4/3-approximation algorithm for finding decompositions of general permutations.

3. Similarity distances on Y-trees: the single cycle case

The gist of the proposed approach for computing the similarity distance on a Y-tree is to decompose a cycle in such a way that all its components are supported on paths. Once such a decomposition is performed, we can invoke the results of our companion paper [6], which asserts that cycle decompositions for metric-path costs can be performed optimally in linear time. The key question is hence to determine if one can perform a decomposition of an arbitrary cycle into cycles that are supported on paths in an efficient manner, i.e., by only using efficient transpositions. For this purpose, we find the following lemma that applies to *general permutations* useful.

Lemma 1. *Let φ be a metric-tree weight function, and let π be a permutation. The minimum decomposition cost of π is bounded below by one half of its displacement, i.e.,*

$$\delta(\pi) \geq \frac{1}{2} D(\pi).$$

The lower bound may be achieved for metric-path weight functions φ, for which

$$\delta(\pi) = \frac{1}{2} D(\pi). \tag{1}$$

The proof of the previous lemma can be found in our companion paper [6], with the latter claim following by induction on the number of elements in the support of the permutation π.

An algorithm which describes how to find a minimum cost decomposition T^* in this case can be easily devised using the idea behind the proof, and is presented next.

Without loss of generality, label the vertices in the defining path from left to right as $1, 2, \ldots, n$, and suppose that we are decomposing a single cycle $\pi = (v_1\, v_2\, \ldots\, v_{|\pi|})$, with $v_1 = \min\ \mathrm{supp}(\pi)$. If this is not the case, rewrite π by cyclically shifting its elements. Let $v_t = \min_{i \in \mathrm{supp}(\pi)} \{i : i \neq v_1\}$. With this notation at hand, the steps of the decomposition procedure are listed in Algorithm 1.

At each call of Algorithm 1, the cycle π is rewritten as one of two possible cycle products, depending on whether $v_t = v_{|\mathrm{supp}(\pi)|}$ holds or not. Intuitively, the decomposition breaks cycles using vertices closest to each other, which minimizes the total cost of the transpositions involved. Fig. 3 illustrates the two possible cases.

As illustrated by the cycle in Fig. 4, this approach cannot be generalized for Y-tree weight functions. In the example, the total displacement $D((1\,2\,3))$ equals 6, while via exhaustive search one can show that

$$\delta((1\,2\,3)) = 4 \neq \frac{1}{2} D((1\,2\,3)).$$

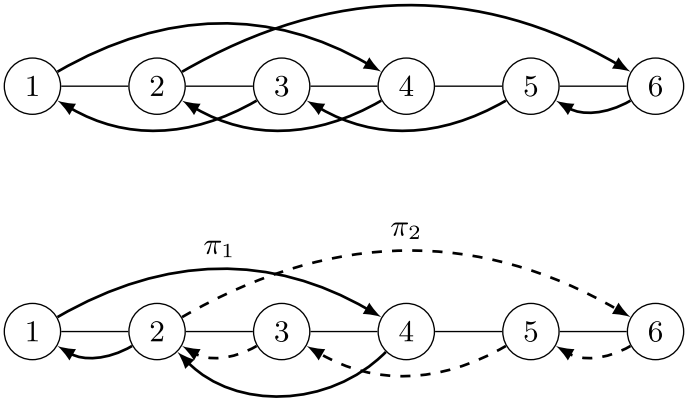
Note that in Fig. 4, the central vertex does not belong to the support of the cycle, and furthermore, the cycle is not balanced. Careful examination hence reveals that in order to generalize Algorithm 1 for Y-tree costs, one has to separately consider three cases: 1) the case when the central vertex belongs to the support of the cycle; 2) the case when the central vertex does not belong to the support of the cycle, but the cycle is balanced; 3) the case when neither of the aforementioned two conditions hold.

We provide next a useful characterization of efficient transpositions. To do so, we recall that the defining graph G is a tree, and that hence there exists a unique path between any two vertices a, b of G. The next lemma describes for which a, b-paths the corresponding transposition $(a\, b)$ is efficient.

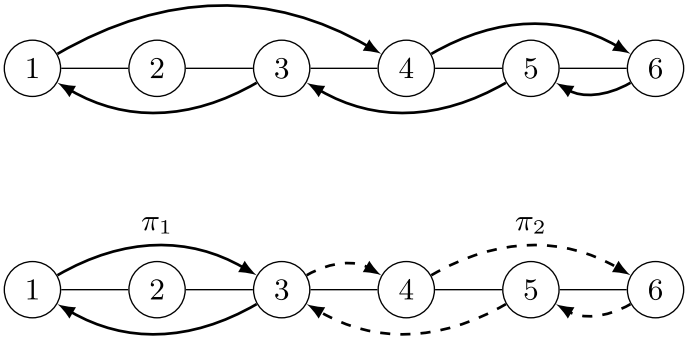
Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.

Algorithm 1: path-td

```
/* Transposition decomposition of cycles for metric-path weights with defining path
   given by 1, ..., n                                                          */
Input: A cycle π = (v_1 v_2 ... v_|π|), with |π| ≥ 2 and v_1 = min supp(π)
Output: A minimum cost decomposition T = (τ_1, ..., τ_|T|) of π, so that π = τ_1 ··· τ_|T|
if |π| = 2 then return (π);
v_t ← min (supp(π)\{v_1});
if v_t ≠ v_|π| then
    π_1 ← (v_1 v_2 ··· v_t);
    π_2 ← (v_t v_{t+1} ··· v_|π|);                                   /* π = π_1 π_2 */
    return Concatenate(path-td(π_1), path-td(π_2));
else
    π_1 ← (v_2 v_3 ··· v_|π|);
    τ ← (v_1 v_|π|);                                                /* π = π_1 τ */
    return Concatenate(path-td(π_1), τ);
end
```



(1) We have $v_t = 2$ and $v_{|\pi|} = 3$, so that $v_t \neq v_{|\pi|}$. Hence, the cycle $(1\,4\,2\,6\,5\,3)$ is decomposed as $(1\,4\,2\,6\,5\,3) = \pi_1\pi_2 = (1\,4\,2)(2\,6\,5\,3)$. In this step of the decomposition, the arc $(3 \to 1)$ is replaced by two arcs, each belonging to one of the cycles. The two resulting cycles are represented with solid and dashed arcs, respectively.



(2) We have $v_t = v_{|\pi|} = 3$. Hence, the cycle $(1\,4\,6\,5\,3)$ is decomposed as $(1\,4\,6\,5\,3) = \pi_1\pi_2 = (3\,4\,6\,5)(1\,3)$. In this step of the decomposition, the arc $(1 \to 4)$ is replaced by two arcs each belonging to one of the component cycles. The resulting two cycles are represented with solid and dashed arcs, respectively.

Fig. 3. Two different decomposition cases encountered in Algorithm 1, based on whether $v_t = v_{|\pi|}$ holds or not.

Lemma 2. *Let G be the defining graph of a metric-tree weight function φ and let π be an arbitrary permutation of length n. For distinct $a, b \in [n]$, we have*

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.

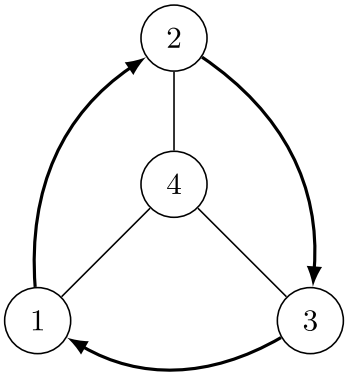


Fig. 4. The weight φ is defined via a Y-tree with all edges of weight one; the cycle equals (1 2 3).

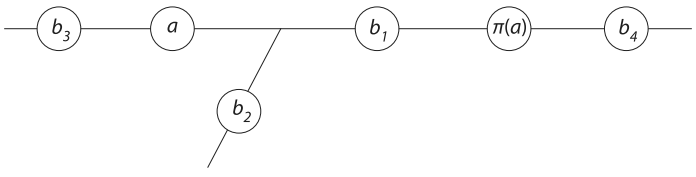


Fig. 5. There are four possible positions for b with respect to a and $\pi(a)$ shown by b_1, b_2, b_3, and b_4.

$$\begin{aligned}\Delta(a, b; \pi) &\geq \varphi(a, b) + \varphi(b, \pi(a)) - \varphi(a, \pi(a)) \geq 0,\\ \Delta(a, b; \pi) &\geq \varphi(a, b) + \varphi(a, \pi(b)) - \varphi(b, \pi(b)) \geq 0.\end{aligned} \tag{2}$$

Furthermore, the following claims are equivalent:

i. *The transposition $(a\, b)$ is efficient.*
ii. *It holds that*

$$\varphi(a, \pi(a)) - \varphi(b, \pi(a)) = \varphi(a, b), \tag{3}$$
$$\varphi(b, \pi(b)) - \varphi(a, \pi(b)) = \varphi(a, b). \tag{4}$$

iii. *The vertex a lies on the $(b, \pi(b))$-path in G, and the vertex b lies on the $(a, \pi(a))$-path in G.*

Proof. First, note that

$$\begin{aligned}\Delta(a, b; \pi) &= 2\varphi(a, b) - D(\pi) + D(\pi\ (a\, b))\\ &= 2\varphi(a, b) - \varphi(a, \pi(a)) - \varphi(b, \pi(b)) + \varphi(a, \pi(b)) + \varphi(b, \pi(a)).\end{aligned} \tag{5}$$

From the triangle inequality, one also has

$$\varphi(a, b) + \varphi(b, \pi(a)) - \varphi(a, \pi(a)) \geq 0, \tag{6}$$
$$\varphi(a, b) + \varphi(a, \pi(b)) - \varphi(b, \pi(b)) \geq 0. \tag{7}$$

By adding (6) and (7), and by using (5), one can show that (2) holds as well. Additionally, $\Delta(a, b; \pi) = 0$ if and only if (6) and (7) hold with equality, that is, if and only if (4) and (5) are true. This proves (2), as well as that claims *i* and *ii* are equivalent.

To show that claims *ii* and *iii* are equivalent, it suffices to show that $\varphi(a, \pi(a)) - \varphi(b, \pi(a)) = \varphi(a, b)$ if and only if b is on the path from a to $\pi(a)$. This can be readily verified by inspecting all possible vertex placements as shown in Fig. 5, and by noting that all weights on the tree are positive. Note that in Fig. 5, we have ignored the case where a, b, and $\pi(a)$ are not all distinct, as this case is particularly simple to check. □

The next lemma strengthens the results of Lemma 1, and will be of use in the derivations to follow.

Lemma 3. *For a permutation π, the gap between $\delta(\pi)$ and $\frac{1}{2}D(\pi)$ equals the sum of the inefficiencies of the transpositions in a minimum weight decomposition.*

Proof. Let $T^* = (\tau_1, \ldots, \tau_{|T^*|})$, $\tau_j = (a_j\, b_j)$, be a minimum weight sorting and let $\pi_j = \pi_{j-1}\tau_j$, with $\pi_0 = \pi$. For all j, we have

$$\varphi(a_j, b_j) - \frac{D(\pi_{j-1}) - D(\pi_j)}{2} = \Delta(a_j, b_j; \pi_{j-1}).$$

By summing over all j, we find

$$\delta(\pi) - \frac{1}{2}D(\pi) = \sum_{j=1}^{|T^*|} \frac{1}{2}\Delta(a_j, b_j; \pi_{j-1}), \tag{8}$$

which produces the desired result. □

Note that by Lemma 2, the right side of (8) is always nonnegative.

Our algorithmic solution to the decomposition problem conceptually consists of two stages. In the first stage, a cycle is represented by a product of shorter adjacent cycles, each of which has the property that its support lies on a path in the Y-tree. In the second stage, we decompose the path-supported cycles using Algorithm 1.

3.1. Case 1: cycles containing the central vertex

We start with an analysis of the decomposition algorithm for the case that the central vertex is contained in a cycle of the permutation under consideration. As before, we denote the central vertex by v_c, and with slight abuse of notation, use B_1, B_2 and B_3 to denote both the three branches of the Y-tree and their corresponding vertex sets. Recall that the central vertex does not belong to any of the branches.

The decomposition procedure for this cycle type is described in Algorithm 2. The algorithm terminates when all subcycles π_j have supports that lie on paths of the defining Y-tree. One step of the decomposition procedure in Algorithm 2 is shown in Fig. 6.

Algorithm 2: central-td

Input: A cycle $\pi = (v_c\ v_1\ \cdots\ v_{|\pi|-1})$ containing the central vertex v_c
Output: A minimum cost decomposition of π

if supp(π) *is contained in a path of the Y-tree* **then return** `path-td`(π);
$t \leftarrow \min\{i \in [|\pi|-1] : v_i \in$ `Br`(v_1), $v_{i+1} \notin$ `Br`$(v_1)\}$;
$\pi' \leftarrow (v_c\ v_{t+1}\ \cdots\ v_{|\pi|-1})$;
$\pi'' \leftarrow (v_c\ v_1\ \ldots\ v_t)$; /* $\pi = \pi'\pi''$ */
return `Concatenate`(`central-td`(π'), `path-td`(π''))

Lemma 4. *Let π_1 and π_2 be two permutations such that* supp$(\pi_1) \cap$ supp$(\pi_2) = \{a\}$. *The following are equivalent:*

1. $D(\pi_1\pi_2) = D(\pi_1) + D(\pi_2)$,
2. *The vertex a lies on the $(\pi_1(a), \pi_2^{-1}(a))$-path.*

If the above conditions hold and, additionally, $\delta(\pi_1) = \frac{1}{2}D(\pi_1)$ and $\delta(\pi_2) = \frac{1}{2}D(\pi_2)$, then $\delta(\pi_1\pi_2) = \frac{1}{2}D(\pi_1\pi_2)$.

Proof. Since supp$(\pi_1) \cap$ supp$(\pi_2) = \{a\}$, we have

$$D(\pi_1\pi_2) - D(\pi_1) - D(\pi_2) = \varphi(\pi_2^{-1}(a), \pi_1(a)) - \varphi(a, \pi_2^{-1}(a)) - \varphi(a, \pi_1(a)).$$

Condition 1 holds if and only if the right side of this equation is 0. By the same reasoning used in Lemma 2, the right side of this equation is 0 if and only if Condition 2 holds.

For the second part, Lemma 1 yields the lower bound $\delta(\pi_1\pi_2) \geq \frac{1}{2}D(\pi_1\pi_2)$, while the hypotheses give the upper bound:

$$\delta(\pi_1\pi_2) \leq \delta(\pi_1) + \delta(\pi_2) = \frac{1}{2}D(\pi_1) + \frac{1}{2}D(\pi_2) = \frac{1}{2}D(\pi_1\pi_2). \quad \square$$

Lemma 5. *The minimum decomposition cost of a cycle π containing the central vertex equals one half of its displacement, i.e.,*

$$\delta(\pi) = \frac{1}{2}D(\pi).$$

Proof. We use induction on $|\pi|$. The smallest (non-trivial) cycle π that contains the central vertex v_c is of the form $(v_c\ b)$ for some b and has only two vertices. Thus

$$\delta(\pi) = \varphi(v_c, b) = \frac{1}{2}D(\pi).$$

Now suppose for any cycle of size at most $m-1$, the lemma holds. We show that it also holds for a cycle π of size m. Algorithm 2 finds two cycles π' and π'' such that $\pi = \pi'\pi''$, supp$(\pi') \cap$ supp$(\pi'') = \{v_c\}$, and the cycle π'' lies on a path of the Y-tree.

Since v_t and v_{t+1} lie on different branches, v_c lies on the unique v_t, v_{t+1}-path. Since π' has size at most $m-1$, the induction hypothesis yields $\delta(\pi') = \frac{1}{2}D(\pi')$, while Lemma 1 yields $\delta(\pi'') = \frac{1}{2}D(\pi'')$. By Lemma 4, we conclude that $\delta(\pi) = \frac{1}{2}D(\pi)$. □

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.

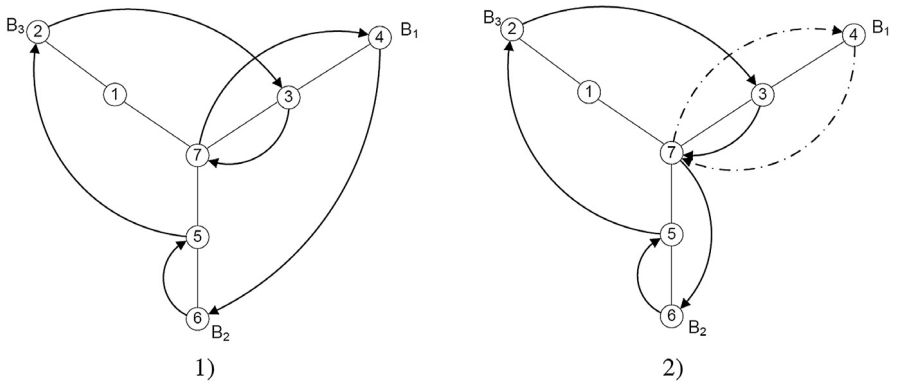


Fig. 6. (1) A Y-tree and input cycle (7 4 6 5 2 3). After the first iteration of Algorithm 2, the arc (4 → 6) is replaced by two arcs, decomposing the original cycle (7 4 6 5 2 3) into a product of two adjacent cycles, i.e., (7 4 6 5 2 3)=(7 6 5 2 3)(7 4), as shown in (2).

3.2. Case 2: balanced cycles

Given that a cycle containing the central vertex was analyzed in Case 1 of our exposition, we henceforth tacitly assume that the balanced cycles considered in this section do not contain the central vertex.

Lemma 6. *For a balanced cycle π, the minimum decomposition cost of π equals one half of its displacement, i.e.,*

$$\delta(\pi) = \frac{1}{2}D(\pi).$$

Proof. We prove the lemma by induction on $|\pi|$.

Clearly, the lemma holds for $|\pi| = 2$. We therefore assume next that it also holds for all cycles κ with $|\kappa| < m$. We then show that the claimed result also holds for π, where π is an arbitrary cycle such that $|\pi| = m$.

Let $\pi = (v_1 \dots v_m)$, and without loss of generality, let the support of π span all three branches of the Y-tree (if the support of the cycle were to lie on two branches only, the desired result would immediately follow from (1)). Consider two distinct indices t and l modulo m, such that v_t and v_{l+1} belong to the same branch, say B_1, with $v_{t+1}, \dots, v_l$ belonging to a different branch, say B_2. Such indices t and l must exist since the cycle π is balanced.

We consider two cases, depending on which one of the two vertices v_t and v_{l+1} is closer to the center v_c. First, suppose v_t is closer to v_c, that is, $\varphi(v_c, v_t) < \varphi(v_c, v_{l+1})$. An illustrative example is shown in Fig. 7.1. Let $\pi' = (v_1 \cdots v_{t-1} v_t v_{l+1} v_{l+2} \cdots v_m)$ and $\pi'' = (v_t \cdots v_l)$. Note that $\pi = \pi'\pi''$, that $\text{supp}(\pi') \cap \text{supp}(\pi'') = \{v_t\}$, and that π' is balanced while π'' lies on a path (see Fig. 7.2). The induction hypothesis yields $\delta(\pi') = \frac{1}{2}D(\pi')$, while Lemma 5 yields $\delta(\pi'') = \frac{1}{2}D(\pi'')$. Since v_t lies on the v_l, v_{l+1}-path, Lemma 4 yields $\delta(\pi) = \frac{1}{2}D(\pi)$.

Next, suppose that $\varphi(v_c, v_t) > \varphi(v_c, v_{l+1})$, as illustrated in Fig. 8. In this case, let $\pi' = (v_1 \cdots v_{t-1} v_t v_{l+1} v_{l+2} \cdots v_m)$ and let $\pi'' = (v_{t+1} \cdots v_{l+1})$. Now $\pi = \pi''\pi'$, with π' lying on a path and with π'' balanced, so we again have $\delta(\pi') = \frac{1}{2}D(\pi')$ and $\delta(\pi'') = \frac{1}{2}D(\pi'')$. Since $\text{supp}(\pi') \cap \text{supp}(\pi'') = \{v_{l+1}\}$ and v_{l+1} lies on the v_t, v_{t+1}-path, Lemma 4 again yields $\delta(\pi) = \frac{1}{2}D(\pi)$. □

Based on the proof of Lemma 6, we present Algorithm 3, which describes the steps for finding a minimum cost decomposition for a balanced cycle π. We use a push-down stack data structure, with the standard push, pop, and peek operations, to search for indices t and l with the properties described in the proof of the above lemma. The stack is denoted by S.

We follow the closed walk induced by π, starting from an arbitrary vertex[3] in the support of the cycle until encountering a branch-changing arc. Such an arc is pushed into the stack S. We keep following the closed walk while pushing arcs in or out of the stack S. Only branch-changing arcs may be added to the stack. Once a branch-changing arc in the "opposite branch direction" of the arc at the top of the stack is encountered, the two arcs are paired up and removed from the stack. The paired arcs dictate the choice of the transpositions (v_t, v_{t+1}) and (v_l, v_{l+1}) in the proof of the previous result, and are used to decompose the current cycle. The procedure is repeated until all the vertices of the cycle are visited exactly once. As each vertex is visited once, the running time of the algorithm is linear in $|\pi|$.

[3] Although the procedure works for an arbitrarily chosen vertex, for ease of demonstration, in Algorithm 3, we simply fix the initial vertex.

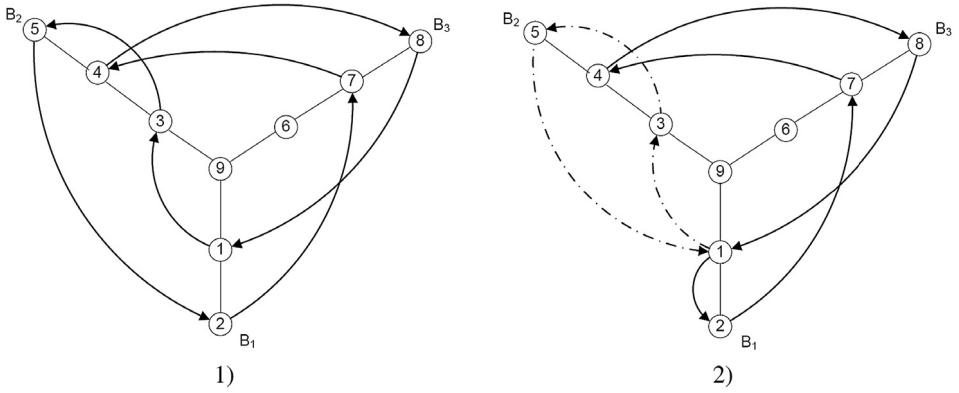


Fig. 7. An example of the decomposition procedure when $\varphi(v_c, v_t) < \varphi(v_c, v_{l+1})$. The cycle equals $\pi = (1\,3\,5\,2\,7\,4\,8)$, with $v_1 = 1 \in B_1$. The first visited arc between branches is $(1 \to 3)$, i.e., $v_t = 1 \in B_1,\ v_{t+1} = 3 \in B_2$; the second visited arc between branches is $(5 \to 2)$, i.e., $v_l = 5 \in B_2,\ v_{l+1} = 2 \in B_1$. As $v_{l+1} = 2 \in B_1$, we decompose $(1\,3\,5\,2\,7\,4\,8)$ into two shorter cycles $(1\,2\,7\,4\,8)$ and $(1\,3\,5)$, i.e., $(1\,3\,5\,2\,7\,4\,8)$=$(1\,2\,7\,4\,8)(1\,3\,5)$.

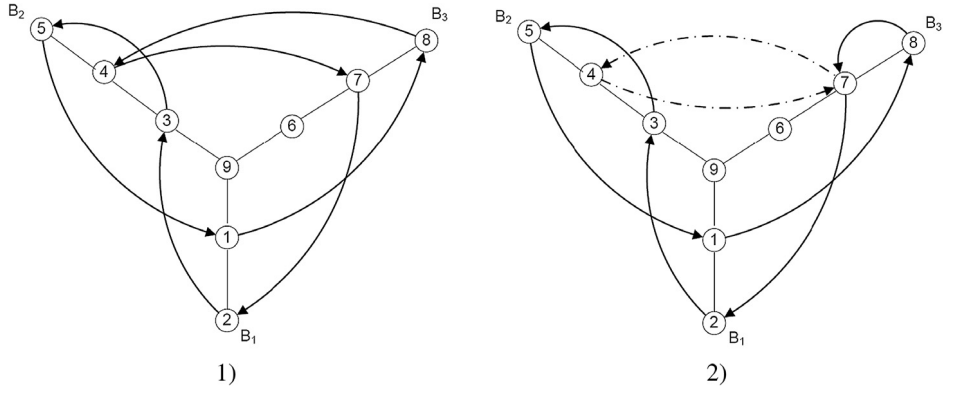


Fig. 8. An example of the decomposition procedure when $\varphi(v_c, v_t) > \varphi(v_c, v_{l+1})$. The cycle equals $\pi = (1\,8\,4\,7\,2\,3\,5)$, with $v_1 = 1 \in B_1$. The first visited arc between branches is $(1 \to 8)$, i.e., $v_t = 1 \in B_1,\ v_{t+1} = 8 \in B_3$; the second visited arc between branches is $(8 \to 4)$, i.e., $v_l = 8 \in B_3,\ v_{l+1} = 4 \in B_2$. As $v_{l+1} = 4 \notin B_1$, we add $(8 \to 4)$ to the stack S and move on to the arc $(4 \to 7)$. Since $v_{l+1} = 7 \in B_3$, we decompose $(1\,8\,4\,7\,2\,3\,5)$ into two shorter cycles $(4\,7)$ and $(1\,8\,7\,2\,3\,5)$, i.e., $(1\,8\,4\,7\,2\,3\,5)$=$(4\,7)(1\,8\,7\,2\,3\,5)$.

3.3. Case 3: unbalanced cycles

In this section, we will determine $\delta(\pi)$ in the case where π is an unbalanced cycle. The proof relies on a lower bound for $\delta(\pi)$ for general permutations π, which we show to hold with equality when π is an unbalanced cycle. To prove this lower bound, we show that every permutation has a min-cost sorting with a particularly useful technical property. We first prove a few smaller lemmas.

Lemma 7. *Let α and β be permutations. If* $\mathrm{supp}(\alpha) \cap \mathrm{supp}(\beta\alpha) = \emptyset$, *then* $\alpha\beta = \beta\alpha$.

Proof. Since $\beta\alpha(x) = x$ for all $x \in \mathrm{supp}(\alpha)$, we see that $\beta = \alpha^{-1}$ on $\mathrm{supp}(\alpha)$. Thus, we can write $\beta = \gamma\alpha^{-1}$ where $\mathrm{supp}(\gamma) \cap \mathrm{supp}(\alpha) = \emptyset$. Since α commutes with both γ and α^{-1}, we see that α commutes with β. □

Lemma 8. *Let $\sigma_1, \ldots, \sigma_p$ be transpositions, and let $\rho = \sigma_1 \cdots \sigma_p$. If for all $i \in [p]$ we have* $\mathrm{supp}(\sigma_i) \cap \mathrm{supp}(\sigma_i\sigma_{i+1}\sigma_{i+2} \cdots \sigma_{i+(p-1)}) = \emptyset$, *where subscripts are taken modulo p, then ρ is the identity permutation.*

Proof. Repeatedly applying Lemma 7 shows that for all i,

$$\rho = \sigma_i\sigma_{i+1} \cdots \sigma_{i+p-1},$$

again with subscripts modulo p. Thus, by hypothesis, we have $\mathrm{supp}(\rho) \cap \mathrm{supp}(\sigma_i) = \emptyset$ for all i. Since $\mathrm{supp}(\rho) \subset \mathrm{supp}(\sigma_1) \cup \cdots \cup \mathrm{supp}(\sigma_p)$, it follows that $\mathrm{supp}(\rho) = \emptyset$. □

We now state and prove our min-cost sorting lemma.

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.

Algorithm 3: balanced-td

Input: A balanced cycle $\pi = (v_1\, v_2 \cdots v_{|\pi|})$
Output: A minimum cost decomposition of π

$S \leftarrow \emptyset, a \leftarrow v_1$; /* a denotes the last vertex visited */
$T \leftarrow \pi$
/* T will be an adjacent cycle decomposition $T = \pi_1, \ldots, \pi_k$ of π (i.e. $\pi = \pi_1 \cdots \pi_k$), where $\mathrm{supp}(\pi_j)$, $j \in [k]$ is contained in a path of the Y-tree */
while π *spans all three branches* **do**
  $j \leftarrow \min\{i : \pi^{i+1}(a) \notin \mathrm{Br}(a)\}$;
  $c_1 \leftarrow \pi^j(a), c_2 \leftarrow \pi(c_1)$; /* $c_1 \to c_2$ is the first unvisited arc that leaves $\mathrm{Br}(a)$ */
  $a \leftarrow c_2$; /* update last visited vertex */
  if $S = \emptyset$ **then**
    $\mathtt{push}(S, c_1 \to c_2)$;
  else
    $(b_1 \to b_2) \leftarrow \mathtt{peek}(S)$;
    if $c_2 \in \mathrm{Br}(b_1)$ **then**
      if $\varphi(v_c, b_1) < \varphi(v_c, c_2)$ **then**
        $\pi'' \leftarrow (b_1 \cdots c_1)$ and $\pi' \leftarrow \pi(\pi'')^{-1}$;
        In T, replace π with the pair π', π'' ; /* the cycle π is decomposed into two cycles */
      else
        $\pi'' \leftarrow (b_2 \cdots c_2)$ and $\pi' \leftarrow (\pi'')^{-1}\pi$;
        In T, replace π with the pair π'', π';
      end
      $\mathtt{pop}(S)$;
      $\pi \leftarrow \pi'$;
    else
      $\mathtt{push}(S, c_1 \to c_2)$;
    end
  end
end
$U \leftarrow ()$;
for $\kappa \in T$ **do**
  $U \leftarrow \mathtt{Concatenate}(U, \mathtt{path\text{-}td}(\kappa))$;
end
return U;

Lemma 9. *Every permutation π has a minimum-cost sorting $(\tau_1, \ldots, \tau_k)$ such that for all i,*

$$\mathrm{supp}(\tau_i) \cap \mathrm{supp}(\tau_i \tau_{i+1} \cdots \tau_n) \neq \emptyset.$$

Proof. Let $(\tau_1, \ldots, \tau_k)$ be a minimum-cost sorting of π. We will show that there is some $\eta \in S_k$ such that $(\tau_{\eta(1)}, \tau_{\eta(k)})$ is a sorting of π with the desired support property. Clearly, any such sorting is also a minimum-cost sorting.

We define an algorithm to manipulate the sorting, and write τ_i to refer to the transposition *currently* in the ith position of the sorting. Say that a transposition τ_i is *bad* in the current sorting if $\mathrm{supp}(\tau_i) \cap \mathrm{supp}(\tau_i \cdots \tau_k) = \emptyset$. Our goal is to permute the transpositions so that there is no bad transposition. Consider the following algorithm:

Algorithm 4: fix-sorting

Input: A min-cost sorting $(\tau_1, \ldots, \tau_k)$ of π.
Output: A min-cost sorting of π with no bad transpositions.
while *there is a bad transposition* **do**
  $B \leftarrow \min\{i : \tau_i \text{ is bad}\}$;
  Move τ_B to the end of the sorting, keeping all other transpositions in the same relative order;
end

By Lemma 7, if τ_B is bad before we execute Step 3, then

$$\tau_B \tau_{B+1} \cdots \tau_k = \tau_{B+1} \cdots \tau_k \tau_B,$$

so the product $\tau_1 \cdots \tau_k$ does not change after executing Step 2. Thus, at all times $(\tau_1, \ldots, \tau_k)$ is a minimum-cost sorting of π. If the algorithm terminates, then it yields a minimum-cost sorting of π with no bad transpositions, as desired. We now show that the algorithm terminates.

Let B_i denote the index B chosen in Step 2 on the ith iteration of the algorithm. The key observation is that the sequence $B_1, B_2, \ldots$ is nondecreasing: if B is the index of the leftmost bad transposition and $C < B$, then by Lemma 7, the rearrangement in Step 2 does not alter the product $\tau_B \cdots \tau_k$, so τ_C remains good after Step 2. Thus, if the algorithm does not terminate, then the sequence B_i is eventually constant: the algorithm is repeatedly choosing the same index B. Let $(\tau_1, \ldots, \tau_k)$ be the current sorting when we first choose the index B, and let $\sigma_1, \ldots, \sigma_p = \tau_B, \ldots, \tau_k$. Since the index B is bad for the rest of the algorithm's run, we have

$$\operatorname{supp}(\sigma_i) \cap \operatorname{supp}(\sigma_i \sigma_{i+1} \cdots \sigma_{i+(p-1)}) = \emptyset$$

for all $i \in [p]$, where indices are taken modulo p. By Lemma 8, it follows that $\sigma_1 \cdots \sigma_p = e$. Now $(\tau_1, \ldots, \tau_{B-1})$ is a lower-cost sorting of π, contradicting the assumption that $(\tau_1, \ldots, \tau_k)$ is a minimum-cost sorting. □

We are now ready to prove the main result of the section.

Lemma 10. *For an unbalanced permutation π, we have*

$$\delta(\pi) \geq \frac{1}{2} D(\pi) + \min_{v_i \in \operatorname{supp}(\pi)} \varphi(v_c, v_i). \tag{9}$$

Furthermore, if π has only one non-trivial cycle, then equality holds.

Proof. To prove Lemma 10, we first derive the lower bound (9), which we subsequently show in a constructive manner to be achievable. Intuitively, the bound suggests that one should first merge the central vertex into the cycle via a smallest cost transposition and then decompose the newly formed cycle. Despite the apparent simplicity of the claim, the proof of the result is rather technical.

Let $T^* = (\tau_1, \ldots, \tau_{|T^*|})$ be a minimum cost sorting of π satisfying the conclusion of Lemma 9. Define $\pi_j = \pi_{j-1}\tau_j$, for all $1 \leq j \leq |T^*|$, with $\pi_0 = \pi$. For all j in $\{1, \ldots, |T^*|\}$, we have $\pi_{j-1} = \tau_{|T^*|} \cdots \tau_j$, so by the choice of π, we have $\operatorname{supp}(\tau_j) \cap \operatorname{supp}(\pi_{j-1}) \neq \emptyset$. Finally, let

$$f_j = \frac{1}{2} \sum_{i=1}^{j} \Delta(\tau_i; \pi_{i-1}) + C(\pi_j),$$

where

$$C(\sigma) = \begin{cases} 0, & \text{if } \sigma \text{ is balanced} \\ \min\limits_{v \in \operatorname{supp}(\sigma)} \varphi(v_c, v), & \text{else} \end{cases}$$

for any permutation σ. Below, we show that f_j is non-decreasing, implying that

$$\min_{v \in \operatorname{supp}(\pi)} \varphi(v_c, v) = f_0 \leq f_{|T^*|} = \frac{1}{2} \sum_{i=1}^{|T^*|} \Delta(\tau_i; \pi_{i-1}) = \delta(\pi) - \frac{1}{2} D(\pi),$$

where the last equality follows from Lemma 3. This proves (9).

To show that f_j is non-decreasing, it suffices to show that

$$\frac{1}{2} \Delta(\tau_j; \pi_{j-1}) \geq C(\pi_{j-1}) - C(\pi_j). \tag{10}$$

If π_{j-1} is balanced or $v_c \in \operatorname{supp}(\pi_{j-1})$, the right side of (10) is non-positive and so (10) holds trivially. Hence, we assume π_{j-1} is unbalanced. There are three cases to consider for π_j: balanced with $v_c \notin \operatorname{supp}(\pi_j)$; $v_c \in \operatorname{supp}(\pi_j)$; and unbalanced. We prove (10) for each case separately:

π_j is balanced with $v_c \notin \operatorname{supp}(\pi_j)$. In this case, since $C(\pi_j) = 0$, we must show

$$\frac{1}{2} \Delta(\tau_j; \pi_{j-1}) \geq \min_{v_i \in \operatorname{supp}(\pi_{j-1})} \varphi(v_c, v_i).$$

The transposition $\tau_j = (a\, b)$ in T^* changes the balance of arcs between two branches. In other words, for some i, k, we have $l_{ik}^{\pi_{j-1}} - l_{ki}^{\pi_{j-1}} \neq l_{ik}^{\pi_j} - l_{ki}^{\pi_j}$. Since the balance of arcs changes, one cannot encounter any of the following placements of the vertices a, b, $a' = \pi_{j-1}(a)$, and $b' = \pi_{j-1}(b)$ on the branches of the Y-tree:

- $\mathrm{Br}[a] = \mathrm{Br}[b]$;
- $\mathrm{Br}[\pi_{j-1}(a)] = \mathrm{Br}[\pi_{j-1}(b)]$;
- $\mathrm{Br}[a] = \mathrm{Br}[\pi_{j-1}(a)]$ and $\mathrm{Br}[b] = \mathrm{Br}[\pi_{j-1}(b)]$;
- $\mathrm{Br}[a] = \mathrm{Br}[\pi_{j-1}(b)]$ and $\mathrm{Br}[b] = \mathrm{Br}[\pi_{j-1}(a)]$.

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.

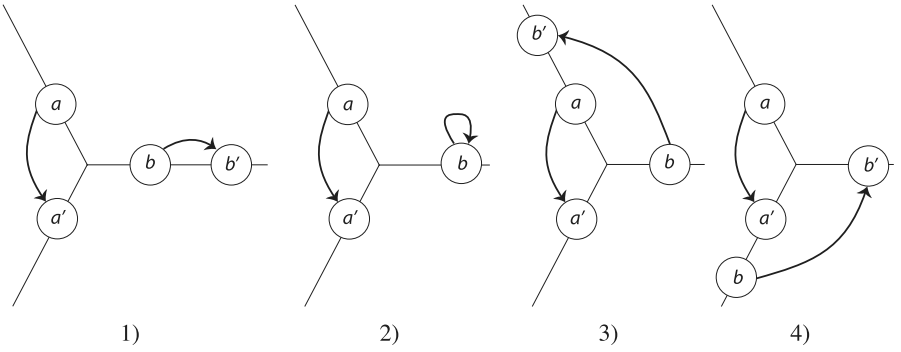


Fig. 9. Illustration for the proof of Lemma 10. Depicted are the configurations for a, b, $a' = \pi_{j-1}(a)$ and $b' = \pi_{j-1}(b)$ that change the balance between branches.

Since $\mathrm{Br}[a] \neq \mathrm{Br}[\pi_{j-1}(a)]$ or $\mathrm{Br}[b] \neq \mathrm{Br}[\pi_{j-1}(b)]$, by symmetry, we may assume $\mathrm{Br}[a] \neq \mathrm{Br}[\pi_{j-1}(a)]$. The cases not covered in the previous list satisfying $\mathrm{Br}[a] \neq \mathrm{Br}[\pi_{j-1}(a)]$ are shown in Fig. 9. Note that in the figure, the exact ordering of vertices on the same branch is irrelevant.

For cases (1), (2) and (3), we have

$$\varphi(a, b) + \varphi(b, \pi_{j-1}(a)) - \varphi(a, \pi_{j-1}(a)) = 2\varphi(b, v_c) \geq 2 \min_{v_i \in \mathrm{supp}(\pi_{j-1})} \varphi(v_c, v_i),$$

and for case (4), we have

$$\varphi(a, b) + \varphi(a, \pi_{j-1}(b)) - \varphi(b, \pi_{j-1}(b)) = 2\varphi(a, v_c) \geq 2 \min_{v_i \in \mathrm{supp}(\pi_{j-1})} \varphi(v_c, v_i).$$

Hence, by (2), it follows that

$$\frac{1}{2}\Delta(a, b; \pi_{j-1}) \geq \min_{v_i \in \mathrm{supp}(\pi_{j-1})} \varphi(v_c, v_i).$$

π_j **contains the central vertex, i.e.,** $v_c \in \mathrm{supp}(\pi_j)$. In this case, since $C(\pi_j) = 0$, we must show

$$\frac{1}{2}\Delta(\tau_j; \pi_{j-1}) \geq \min_{v_i \in \mathrm{supp}(\pi_{j-1})} \varphi(v_c, v_i).$$

Since π_{j-1} is unbalanced, it does not contain the central vertex. Since $\mathrm{supp}(\pi_j) \subset \mathrm{supp}(\pi_{j-1}) \cup \mathrm{supp}(\tau_j)$, this implies $v_c \in \mathrm{supp}(\tau_j)$. Write $\tau_j = (v_c\, b)$. Since $\mathrm{supp}(\tau_j) \cap \mathrm{supp}(\pi_{j-1}) \neq \emptyset$, we have $b \in \mathrm{supp}(\pi_{j-1})$. Then, by (2), and the fact that $\pi_{j-1}(v_c) = v_c$,

$$\Delta(v_c, b; \pi_{j-1}) \geq \varphi(v_c, b) + \varphi(b, v_c) - \varphi(v_c, v_c) = 2\varphi(v_c, b) \geq 2 \min_{v_i \in \mathrm{supp}(\pi_{j-1})} \varphi(v_c, v_i).$$

π_j **is unbalanced and** $v_c \notin \mathrm{supp}(\pi_j)$. In this case, we must show

$$\frac{1}{2}\Delta(\tau_j; \pi_{j-1}) \geq \min_{v_i \in \mathrm{supp}(\pi_{j-1})} \varphi(v_c, v_i) - \min_{v_i \in \mathrm{supp}(\pi_j)} \varphi(v_c, v_i). \tag{11}$$

Let $\tau_j = (a\, b)$. If $a, b \in \mathrm{supp}(\pi_{j-1})$, then $\mathrm{supp}(\pi_j) \subseteq \mathrm{supp}(\pi_{j-1})$ and thus

$$\min_{v_i \in \mathrm{supp}(\pi_{j-1})} \varphi(v_c, v_i) \leq \min_{v_i \in \mathrm{supp}(\pi_j)} \varphi(v_c, v_i).$$

Hence the right side of (11) is non-positive and its left side is non-negative, so it holds.

Since $\mathrm{supp}(\tau_j) \cap \mathrm{supp}(\pi_{j-1}) \neq \emptyset$, we cannot have both $a \notin \mathrm{supp}(\pi_{j-1})$ and $b \notin \mathrm{supp}(\pi_{j-1})$. So as the final case, we may assume $a \notin \mathrm{supp}(\pi_{j-1})$ but $b \in \mathrm{supp}(\pi_{j-1})$. We may also assume that $\varphi(a, v_c) < \varphi(v_i, v_c)$ for all $v_i \in \mathrm{supp}(\pi_{j-1})$, since otherwise the right side of (11) is again nonpositive, as $\mathrm{supp}(\pi_j) \subset \mathrm{supp}(\pi_{j-1}) \cup \{a\}$. Since $\pi_{j-1}(a) = a$, applying (2) yields

$$\begin{aligned}
\frac{1}{2}\Delta(\tau_j; \pi_{j-1}) &\geq \frac{1}{2}\left(\varphi(a, b) + \varphi(b, \pi_{j-1}(a)) - \varphi(a, \pi_{j-1}(a))\right) \\
&= \varphi(a, b) \\
&\geq \varphi(b, v_c) - \varphi(a, v_c) \\
&= \varphi(b, v_c) - \min_{v_i \in \mathrm{supp}(\pi_j)} \varphi(v_i, v_c) \\
&\geq \min_{v_i \in \mathrm{supp}(\pi_{j-1})} \varphi(v_i, v_c) - \min_{v_i \in \mathrm{supp}(\pi_j)} \varphi(v_i, v_c).
\end{aligned}$$

Note that the second inequality follows from the triangle inequality applied to a, b, and v_c.

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.

This completes the proof of the fact that f_j is non-decreasing, and the proof of (9). We now show if π has only one (non-trivial) cycle, the lower bound of (9) is achievable.

Consider the cycle $\pi = (v_1 \ \ldots, m\, v_{|\pi|})$ and let v_j be the element of $\mathrm{supp}(\pi)$ that minimizes $\varphi(v_c, v_i)$. There are two cases to consider: either v_j lies on the same branch as v_{j+1}, or it lies on a different branch (see Fig. 10.). If it lies on a different branch, we let $\pi = \pi_1(v_j\, v_c)$, where

$$\pi_1 = (v_1 \ \ldots, m\, v_j\, v_c\, v_{j+1} \ \ldots, m\, v_{|\pi|}). \tag{12}$$

Since π_1 contains the center, $\delta(\pi_1) = \frac{1}{2}D(\pi_1) = \frac{1}{2}D(\pi)$. Hence, $\delta(\pi) \le \frac{1}{2}D(\pi_1) + \varphi(v_c, v_j)$.

If v_j lies on the same branch as v_{j+1}, let

$$k = j - 1 + \min\{h : \pi^h(v_j) \notin \mathrm{Br}(v_j), \}$$

so that v_{k+1} is the closest vertex following v_j in the cycle that does not lie on the same branch as v_j. We then write $\pi = \pi_1\pi_2$, where

$$\begin{aligned} \pi_1 &= (v_1 \ \ldots, m\, v_j\, v_{k+1} \ \ldots, m\, v_{|\pi|}), \\ \pi_2 &= (v_j \ \ldots, m\, v_k). \end{aligned} \tag{13}$$

Note that the cycle π_1 is unbalanced, but v_j and v_{k+1} lie on different branches. Hence, based on the analysis of the previous case, one has

$$\delta(\pi_1) \le \frac{1}{2}D(\pi_1) + \varphi(v_c, v_j).$$

As the support of π_2 is contained in a single branch, Lemma 2 implies that

$$\delta(\pi_2) \le \frac{1}{2}D(\pi_2).$$

Since v_j and v_k lie on the same branch and $\varphi(v_j, v_c) \le \varphi(v_k, v_c)$, we see that v_j lies on the v_k, v_c-path and, hence, the v_k, v_{k+1} path. By Lemma 4, we have $D(\pi) = D(\pi_1) + D(\pi_2)$, so that

$$\begin{aligned} \delta(\pi) &\le \delta(\pi_1) + \delta(\pi_2) \\ &= \frac{1}{2}D(\pi_1) + \varphi(v_c, v_j) + \frac{1}{2}D(\pi_2) \\ &= \frac{1}{2}D(\pi) + \varphi(v_c, v_j). \end{aligned}$$

This completes the proof of the lemma. □

As a final remark, note that the exposition in Sections 3.1–3.3 implicitly assumes that certain properties (such as balancedness) of a specific cycle are known beforehand. However, if this is not the case, additional steps have to be performed to test for such properties, and they reduce to straightforward search and counting procedures. The complexity of this search is linear in the size of the permutation.

Algorithm 5: unbalanced-td

Input: A cycle $\pi = (v_1\, v_2 \cdots v_{|\pi|})$
Output: A minimum cost decomposition of π

$v_j \leftarrow \min_{v_i \in \mathrm{supp}(\pi)} \varphi(v_c, v_i)$;
if *v_j and v_{j+1} lie on different branches* **then**
  return `Concatenate` (`central-td` $((v_1 \cdots v_j\, v_c\, v_{j+1} \cdots v_{|\pi|}))$, $(v_j\, v_c)$);
else
  return `Concatenate` (`unbalanced-td` $((v_1 \cdots v_j\, v_{k+1} \cdots v_{|\pi|}))$, `path-td` $((v_j \cdots v_k))$);
end

We summarize our findings in the following theorem.

Theorem 1. *Let φ be a Y-tree weight function and let π be a cycle permutation. If π does not contain the central vertex and is unbalanced, then*

$$\delta(\pi) = \frac{1}{2}D(\pi) + \min_{v_i \in \mathrm{supp}(\pi)} \varphi(v_c, v_i).$$

Otherwise,

$$\delta(\pi) = \frac{1}{2}D(\pi).$$

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.

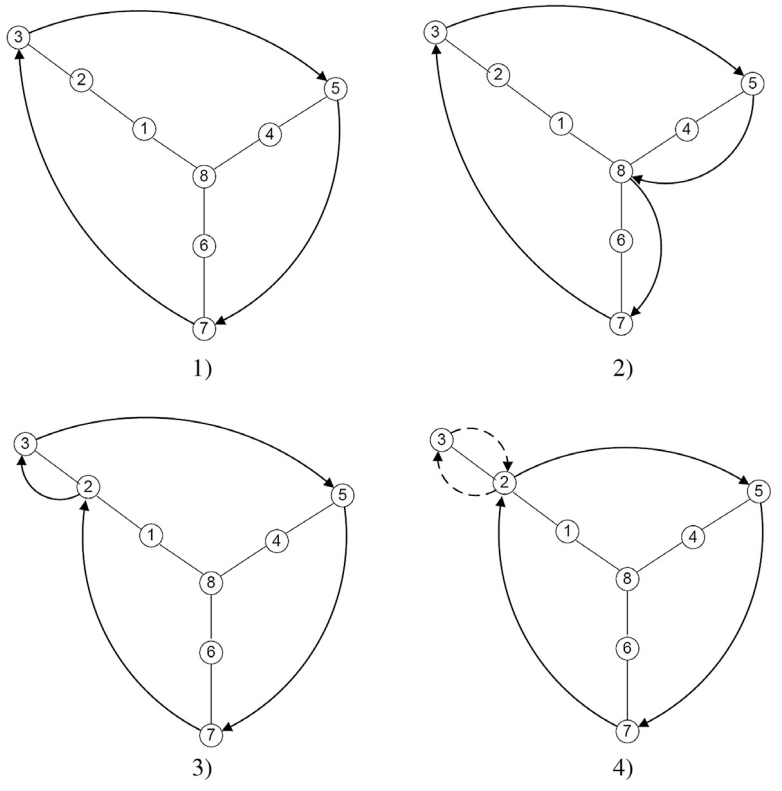


Fig. 10. (1) The cycle $\pi = (3\,5\,7)$. We have $v_j = 5$, and v_j, v_{j+1} lie on different branches; (2) The cycle $\pi_1 = (3\,5\,8\,7)$ (see Eq. (12)); (3) The cycle $\pi = (2\,3\,5\,7)$. We have $v_j = 2$, and v_j, v_{j+1} lie on the same branch; (4) The cycles $\pi_1 = (2\,5\,7)$ and $\pi_2 = (2\,3)$ (see Eq. (13)).

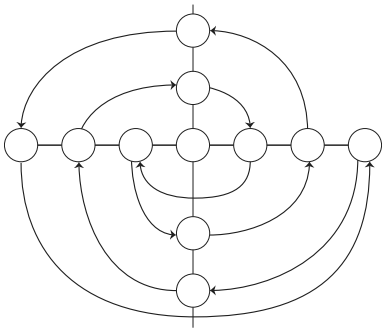


Fig. 11. Example of a balanced cycle on a star tree with four branches that cannot be optimally decomposed using Algorithm 3. The labels of the vertices have no bearing on the finding, and are hence not included.

We conclude this section by noting that it may appear straightforward to extend the results of Algorithms 2, 3, and 4 to a more general defining tree model. This, unfortunately, is not the case even when one shifts from Y-trees, in which the unique node with degree larger than two has degree three, to so called *star-trees*, in which the unique node with degree larger than two may have degree larger than three. In particular, Algorithm 3 cannot be immediately extended as it relies on the fact that a balanced cycle $\pi = (v_1 \dots v_m)$ on a defining Y-tree, there exist two distinct indices t and l modulo m, such that v_t and v_{l+1} belong to the same branch, and $v_{t+1}, \dots, v_l$ belong to a different branch (see proof of Lemma 5). An example of a balanced cycle on a star-tree that does not satisfy this property is shown in Fig. 11.

3.4. Computational complexity of decomposing individual cycles

Careful examination of the algorithms described in the previous sections reveals that three major computational steps are involved in finding a minimum cost decomposition, including: (1) Identifying the type of the cycle; (2) conducting an

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.

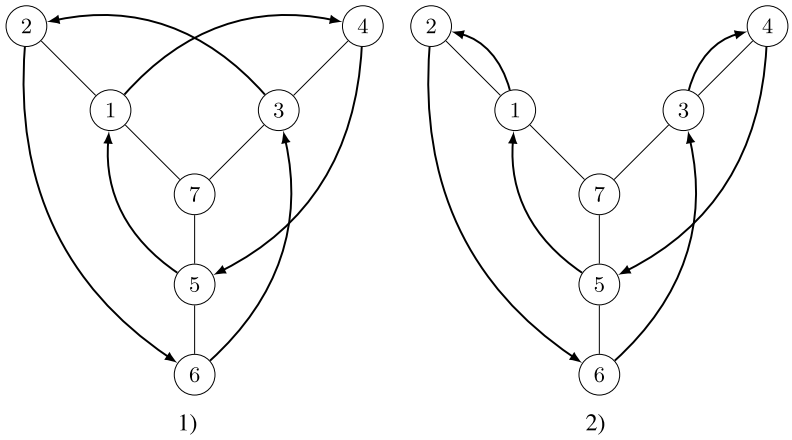


Fig. 12. Merging two cycles creates a balanced cycle: In (1) π is shown as the product of two cycles (1 4 5) and (2 6 3); the merged cycle after applying transposition (1 3) is shown in (2).

adjacent cycle decomposition; (3) solving the individual sub-cycle decomposition problems with supports on paths. From a complexity viewpoint, step (1) requires $O(n)$ operations for checking whether the central vertex v_c belongs to the cycle or not. If the central vertex belongs to the cycle, the decomposition calls for Algorithm 2, which requires $O(n)$ operations. Otherwise, in order to check whether the given cycle is balanced or unbalanced, one has to traverse the cycle to count the number of edges crossing branches and store/compare the values of l_{ij} for all pairs of $i, j \in \{1, 2, 3\}$. This counting procedure requires $O(n)$ operations.

When Algorithm 2 and Algorithm 3 are used, we follow the given cycle and at each vertex we check whether the optimal decomposition conditions are met. Each check requires constant computational time, and as Algorithm 2 and Algorithm 3 terminate when each vertex in the cycle is visited exactly once and when the path decomposition if performed. To solve multiple path cycle decompositions individually, inductive arguments show that at most $m-2$ operations are needed, where m denotes the length of the cycle. In addition, $\sum_{i=1}^{k}|\text{supp}(\pi_i)| = |\pi| + (k-1)$, where k is the number of cycles supported on paths in an adjacent cycle decomposition of π. As a result, since the complexity of both focal steps in the algorithm equals $O(n)$, the overall complexity of the methods equals $O(n)$.

Algorithm 5 proceeds along the same lines as Algorithm 2, except for an additional minimization procedure, which requires $O(n)$ operations. As a result, the complexity of this algorithm also equals $O(n)$.

4. General permutations

Computing the weighted transposition distance between permutations with multiple cycles under the Y-tree weights model is significantly more challenging than computing the same distance between the identity and a single cycle. We currently do not know of any efficient procedure for computing this distance exactly for an arbitrary permutation. Nevertheless, in this section, we describe a straightforward linear-time 4/3-approximation algorithm.

Let us start by recalling a solution to the decomposition problem when all transposition weights are equal: perform the disjoint cycle decomposition and then sort each cycle independently. However, this independent cycle decomposition strategy does not always produce optimal solutions for general weight functions, as illustrated by the example of the permutation $\pi = (4, 6, 2, 5, 1, 3, 7)$ depicted in Fig. 12. Decomposing each cycle of this permutation independently has total cost strictly larger than $\frac{1}{2}D(\pi)$. Alternatively, π may be sorted by first applying the transposition (1 3), thereby merging the cycles (1 4 5) and (2 6 3). As the resulting cycle is balanced, it can be subsequently sorted via a sequence of efficient transpositions. Since the transposition (1 3) is efficient as well, the resulting transform has cost $\delta(\pi) = \frac{1}{2}D(\pi)$. However, even the method of merging cycles may not always be optimal, as may be seen from the example given in Fig. 13.

While decomposing every cycle independently may be in general sub-optimal, the process still provides a 4/3-approximation to an optimal solution. To see this, we first prove that for any cycle κ,

$$\delta(\kappa) \leq \frac{2}{3}D(\kappa). \tag{14}$$

For cycles that lie on a path of the Y-tree, cycles that contain the central vertex, and balanced cycles, this follows from Lemmas 1, 5, and 6, respectively. For an unbalanced cycle κ, from Lemma 10, we have

$$\delta(\kappa) \leq \frac{1}{2}D(\kappa) + \min_{v_i \in \text{supp}(\kappa)} \varphi(v_c, v_i).$$

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.

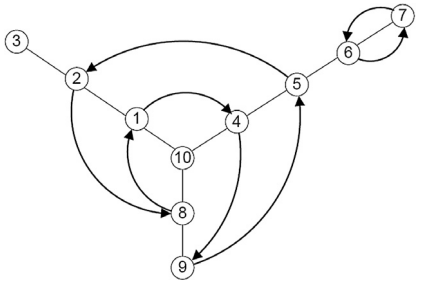


Fig. 13. An example illustrating that merging two cycles may lead to a suboptimal solution: For the permutation $\pi = (4, 8, 3, 9, 2, 7, 6, 1, 5, 10)$, via exhaustive search it can be determined that the minimum decomposition cost equals $\frac{1}{2}D(\pi, e)$ instead of $\frac{1}{2}D(\pi, e) + \varphi(5, 6)$, which may be obtained via merging cycles.

Hence, to show that $\delta(\kappa) \leq \frac{2}{3}D(\kappa)$, it suffices to prove that for an unbalanced cycle κ, $\min_{v_i \in \text{supp}(\kappa)} \varphi(v_c, v_i) \leq \frac{1}{6}D(\kappa)$. Let w_1, w_2, and w_3, given by

$$w_1 = \min_{v_i \in \text{supp}(\kappa) \cap B_1} \varphi(v_c, v_i)$$
$$w_2 = \min_{v_i \in \text{supp}(\kappa) \cap B_2} \varphi(v_c, v_i)$$
$$w_3 = \min_{v_i \in \text{supp}(\kappa) \cap B_3} \varphi(v_c, v_i),$$

be the cost of transposing v_c with the closest element to v_c in $\text{supp}(\kappa)$ on each of the three branches. Without loss of generality, assume that $w_1 = \min_{v_i \in \text{supp}(\kappa)} \varphi(v_c, v_i)$, that is, $w_1 \leq w_2$ and $w_1 \leq w_3$. Since κ is unbalanced, it must contain arcs between all pairs of branches. Thus, since there are at least three arcs,

$$\begin{aligned} D(\kappa) &\geq (w_1 + w_2) + (w_2 + w_3) + (w_3 + w_1) \\ &\geq 2\,(w_1 + w_2 + w_3) \\ &\geq 6w_1 \\ &= 6 \min_{v_i \in \text{supp}(\kappa)} \varphi(v_c, v_i), \end{aligned}$$

which established the desired result. So (14) holds for any cycle κ.

Let π be a permutation with cycles $\kappa_1 \cdots \kappa_m$. If we decompose each cycle κ_i independently of the other cycles using Algorithms 1, 2, 3, and 4, the total cost equals $\sum_{i=1}^{m} \delta(\kappa_i)$. This leads to

$$\sum_{i=1}^{m} \delta(\kappa_i) \leq \frac{2}{3} \sum_{i=1}^{m} D(\kappa) = \frac{2}{3} D(\pi) \leq \frac{4}{3} \delta(\pi), \tag{15}$$

where the first inequality follows from (14) and the second inequality follows form Lemma 1. Hence, the approximation factor is $4/3$, as claimed.

As a final remark, we would like to point out that the unbalanced cycles may be merged according to their lengths in order to provide practical improvements to the theoretical approximation bound of $4/3$. The procedure asks for merging the central vertex v_c with the unbalanced cycle of longest length, m, by using a vertex in its support closest to v_c. Given that there are m arcs,

$$D(\kappa) \geq (2m) \min_{v_i \in \text{supp}(\kappa)} \varphi(v_c, v_i).$$

Once the central vertex v_c is included in the newly formed cycle, one can merge other unbalanced cycles into the cycle via smallest cost transpositions involving the central vertex. In this case, the approximation constant equals $1 + 1/m$.

We summarized the results regarding decomposition for metric-graph weights in Table 1. For path and Y-tree cases in the table, we provide decomposition algorithms that have complexity $O(n)$. The problem is open for more general graphs.

5. Conclusion and open problems

We introduced the notion of similarity distance between rankings under Y-tree weights and presented a polynomial-time algorithm for computing the distance between cycle permutations in terms of the displacement function. The algorithm was centered around the idea of adjacent cycle decomposition, i.e., rewriting a cycle as a product of adjacent/disjoint shorter cycles, where the support of each cycle can be embedded on a path in the defining graph of the Y-tree.

We also described a linear-time decomposition algorithm for permutations that may be embedded in the Y-tree as non-intersecting cycles, and the procedure reduced to finding the shortest path between two non-intersecting cycles. As for

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.

Table 1
Summary of results for sorting/decomposing a permutation π with metric-graph weights.

Graph	Permutation π	Distance $\delta(\pi)$	Source
Tree	General	$\geq \frac{1}{2}D(\pi)$	Lemma 1
Path	General	$= \frac{1}{2}D(\pi)$	Lemma 1
Y-tree	Cycle with central vertex	$= \frac{1}{2}D(\pi)$	Lemma 5
Y-tree	Balanced cycle	$= \frac{1}{2}D(\pi)$	Lemma 6
Y-tree	Unbalanced permutation	$\geq \frac{1}{2}D(\pi) + \min_{v_i \in \mathrm{supp}(\pi)} \varphi(v_c, v_i)$	Lemma 10
Y-tree	Unbalanced cycle	$= \frac{1}{2}D(\pi) + \min_{v_i \in \mathrm{supp}(\pi)} \varphi(v_c, v_i)$	Lemma 10
Y-tree	General	$\geq \frac{3}{4}\sum \delta(\kappa_i)$, κ_i are cycles of π	Eq. (15)

general permutations, we developed a linear time, 4/3-approximation algorithm which is governed by the fact that if there exists an arc emanating from the central vertex that intersects all cycles across branches, then all cycles across branches can be merged efficiently.

There are two main avenues by which the algorithm in this paper could be improved: by finding an *exact* algorithm for finding min-cost decompositions on Y-trees, or by finding an approximation algorithm that works for a larger class of cost functions. It remains unclear whether the problem of finding a min-cost decomposition into transpositions is NP-hard, even when arbitrary cost functions are allowed for the transpositions. Nor is it clear whether the problem lies in NP: if transposition costs are allowed to be exponential in the size of the problem, then it is not clear that the length of a min-cost decomposition is polynomially bounded, as a min-cost decomposition may prefer to use exponentially many "cheap" transpositions rather than a single "expensive" transposition. We close with the following conjecture, which would imply that the problem lies in NP.

Conjecture 1. *Let $\varphi : \binom{[n]}{2} \to \mathbb{R}^+$ be any cost function on the transpositions of S_n. For every permutation $\pi \in S_n$, there is a min-cost decomposition of π using at most $\binom{n}{2}$ transpositions.*

Conjecture 1 has been confirmed by computer search for $n \leq 5$. For each fixed n, verifying Conjecture 1 is a finite computation, since for any fixed path P in the Cayley graph of S_n generated by the transpositions, the cost of the path P is a linear function of the costs of the transpositions, and we can therefore use linear programming to check that there is no cost assignment for which a "long" path in the Cayley graph (i.e., a decomposition using many transpositions) is strictly cheaper than all shorter paths.

The $\binom{n}{2}$ in Conjecture 1 is the best possible bound on the length of a min-cost decomposition. Consider the cost function φ defined by

$$\varphi(i,j) = \begin{cases} 1, & \text{if } |i-j| = 1, \\ n!, & \text{otherwise.} \end{cases}$$

Any min-cost decomposition of a permutation π under this weight function uses only transpositions of the form $(i\, i+1)$. That is, the transpositions used in a min-cost decomposition are exactly the transpositions that are permitted in the bubble sort algorithm, and it is known (see chapter 5 of [15]) that bubble sort uses $\binom{n}{2}$ transpositions in the worst case.

References

[1] Stanislav Angelov, Keshav Kunal, Andrew McGregor, Sorting and selection with random costs, in: LATIN 2008: Theoretical Informatics, Springer, 2008, pp. 48–59.
[2] V. Bafna, P. Pevzner, Sorting by transpositions, SIAM J. Discrete Math. 11 (2) (1998) 224–240.
[3] Moses Charikar, Ronald Fagin, Venkatesan Guruswami, Jon Kleinberg, Prabhakar Raghavan, Amit Sahai, Query strategies for priced information (extended abstract), in: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 582–591 (electronic) MR 2115296.
[4] Moses Charikar, Ronald Fagin, Venkatesan Guruswami, Jon Kleinberg, Prabhakar Raghavan, Amit Sahai, Query strategies for priced information, J. Comput. System Sci. 64 (4) (2002) 785–819 Special issue on STOC 2000 (Portland, OR). MR 1912303.
[5] Persi Diaconis, R.L. Graham, Spearman's footrule as a measure of disarray, J. R. Stat. Soc. Ser. B Stat. Methodol. 39 (2) (1977) 262–268.
[6] Farzad Farnoud (Hassanzadeh), Olgica Milenkovic, Sorting of permutations by cost-constrained transpositions, IEEE Trans. Inform. Theory 58 (1) (2012) 3–23.
[7] Guillaume Fertin, Combinatorics of Genome Rearrangements, MIT Press, 2009.
[8] I.P. Goulden, D.M. Jackson, Combinatorial Enumeration, Dover Publications, 2004.
[9] Anupam Gupta, Amit Kumar, Sorting and selection with structured costs, in: 42nd IEEE Symposium on Foundations of Computer Science (Las Vegas, NV, 2001), IEEE Computer Soc., Los Alamitos, CA, 2001, pp. 416–425 MR 1948730.
[10] Anupam Gupta, Amit Kumar, Where's the winner? Max-finding and sorting with metric costs, in: Approximation, Randomization and Combinatorial Optimization, in: Lecture Notes in Comput. Sci., vol. 3624, Springer, Berlin, 2005, pp. 74–85 MR 2193677.
[11] M. Hofri, Analysis of Algorithms: Computational Methods and Mathematical Tools, Oxford University Press, 1995.
[12] Thomas Huth, Dirk C. Mattfeld, Integration of vehicle routing and resource allocation in a dynamic logistics network, Transp. Res. Part C Emerg. Technol. 17 (2) (2009) 149–162 Selected papers from the Sixth Triennial Symposium on Transportation Analysis (TRISTAN VI).
[13] Sampath Kannan, Sanjeev Khanna, Selection with monotone comparison costs, in: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (Baltimore, MD, 2003), ACM, New York, 2003, pp. 10–17 MR 1974896.

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.

[14] Oren Kapah, Gad M. Landau, Avivit Levy, Nitsan Oz, Interchange rearrangement: the element-cost model, in: Amihood Amir, Andrew Turpin, Alistair Moffat (Eds.), String Processing and Information Retrieval, in: Lecture Notes in Computer Science, vol. 5280, Springer Berlin Heidelberg, 2009, pp. 224–235.
[15] Donald E. Knuth, The Art of Computer Programming. Vol. 3, Addison-Wesley, Reading, MA, 1998, p. xiv+780 Sorting and searching, Second edition [of MR0445948]. MR 3077154.
[16] Ravi Kumar, Sergei Vassilvitskii, Generalized distances between rankings, in: Proceedings of the 19th International Conference on World Wide Web, WWW'10, ACM, New York, NY, USA, 2010, pp. 571–580.
[17] D. Sculley, Rank aggregation for similar items, in: Proceedings of the 7th SIAM International Conference on Data Mining, Citeseer, 2007, pp. 587–592.
[18] Léon-Charles Tranchevent, Roland Barriot, Shi Yu, Steven Van Vooren, Peter Van Loo, Bert Coessens, Bart De Moor, Stein Aerts, Yves Moreau, ENDEAVOUR update: a web resource for gene prioritization in multiple species, Nucleic Acids Res. 36 (suppl 2) (2008) W377–W384.
[19] J.H. van Lint, R.M. Wilson, A Course in Combinatorics, Cambridge University Press, 2001.

Please cite this article in press as: F. Farnoud (Hassanzadeh), et al., Computing similarity distances between rankings, Discrete Applied Mathematics (2017), http://dx.doi.org/10.1016/j.dam.2017.07.038.