Similarity Distances Between Permutations

Lili Su, Farzad Farnoud and Olgica Milenkovic
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign (UIUC), Urbana, IL
Email: {lilisu3,farnoud,milenkov}@illinois.edu

***Abstract*—We address the problem of computing distances between rankings that take into account similarities between elements. The need for evaluating such distances arises in applications such as machine learning, social sciences and data storage. The problem may be summarized as follows: Given two rankings 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 ranking into another. Our focus is on costs that may be described via special tree structures and on rankings modeled as permutations. The presented results include a quadratic-time algorithm for finding a minimum cost transform for a single cycle; and a linear time, $5/3$-approximation algorithm for permutations that contain multiple cycles.**

I. Introduction

Search engines, recommenders, social data aggregation centers and online voting sites are all centered around the task of ranking distinguishable objects according to some predefined criteria [1], [9]. Rankings are frequently provided by different experts or generated based on different search queues, and in order to compare the outputs of the systems, one has to be able to measure the distance between two rankings. The simplest model assumes data in the form of permutations, and the existence of a reference permutation (say, the identity). In this case, the problem of evaluating the similarity between permutations essentially reduces to the problem of sorting.

The problem of sorting distinct elements according to given criteria has a long history, and it was studied in mathematics, computer science, and social choice theory alike. One way to sort permutations is via pairwise comparisons or transpositions of pairs of elements, and many distance functions on permutations account for the number of transpositions needed to convert one permutation into another. Two of the most frequently used such distances are the Cayley metric and the Kendall metric [4], [8], [6], [3], in which each transposition or adjacent transposition, respectively, contributes equally to the overall distance. Only a handful of results are known about specialized sorting problems in which one assigns positive weights or random costs to basic rearrangement steps [2], [7], [5].

The problem of interest may be stated as follows: For a given set of costs assigned to transpositions, find a smallest cost sequence of transpositions (henceforth, termed transform) converting a given permutation to the identity. Our focus is on constraints that take into account that candidates may be similar and that transposing similar candidates induces a smaller cost than transposing dissimilar candidates. We refer to the underlying family of distance measures as *similarity distances*. The here defined similarity distances are not to be confused with the metrics used in [10] that have the goal to ensure that similar items are ranked close to each other in a consensus order.

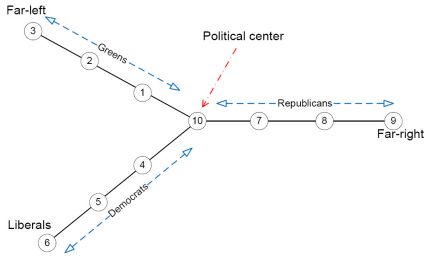

Fig. 1: A "Y-tree" governing the cost of transposing candidates: Swapping candidates closer in their political beliefs induces a smaller contribution to the overall distance.

To illustrate the notion of a similarity distance, consider an example from voting theory. When ranking politicians, one often needs to take into account that candidates come from different parties. Swapping candidates from the same party may be perceived as having a smaller impact on the overall diversity of the outcome of an election than doing otherwise. In this context, assume that one is given the following three rankings of politicians:

$$\sigma_1 = (\text{Clinton, Obama, Bush, Kerry, Romney}),$$
$$\sigma_2 = (\text{Obama, Clinton, Bush, Kerry, Romney}),$$
$$\sigma_3 = (\text{Clinton, Bush, Obama, Kerry, Romney}).$$

Notice that σ_2 and σ_3 differ from σ_1 only in one (adjacent) transposition. In the first case, the swap involves members of the same party, while in the second case, the transposed candidates belong to two different parties. It would hence be reasonable to assume that the distance between σ_1 and σ_2 is smaller than the distance between σ_1 and σ_3. To capture this similarity, we arrange candidates into a tree-structure, where each edge has a certain non-negative weight. The transposition cost of two candidates equals the weight of the unique path between them (see Fig. 1). The tree has only one vertex of degree larger than two, corresponding to the political center, with Republicans, Democrats and Greens placed on different branches of the tree. On this tree, two members of the same party are, on average, closer to each other than two members of different parties. Hence, similarity of candidates may be

978-1-4799-5186-4/14/$31.00 ©2014 IEEE

measured via their distances in the tree. Similar "star shaped" trees arise in phylogenomics, where each branch contains species that evolved from each other, and where similarity is captured at the genomic sequence level.

The contributions of this work are three-fold. First, we introduce Y-tree cost functions and the notion of a similarity distance between permutations. In this setting, the cost of transposing two elements equals the weight of the unique path in the Y-tree, i.e., a tree with largest degree three and exactly one node of degree three. Second, we describe an exact, quadratic-time decomposition algorithm for cycle permutations under Y-tree costs. Third, we develop a linear-time, constant-approximation method for computing the similarity distance between arbitrary permutations. Computing the similarity distance for arbitrary tree structures is extremely difficult and will be discussed elsewhere.

To the best of our knowledge, no prior work on tree-metric costs and sorting was reported in the literature. The results closest in theme to our contribution consider sorting with length-constrained costs and random costs [2], [7].

The paper is organized as follows. Section II contains the mathematical preliminaries and the problem formulation. The similarity distance of cycles is analyzed in Section III. An approximate algorithm for computing the similarity distance between arbitrary permutations is presented in Section IV.

II. Mathematical Preliminaries

A. Notation

Let $[n] \triangleq \{1, 2, \cdots, n\}$. A permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ is a bijection over $[n]$, where $\sigma(i)$ is the image of $i \in [n]$ under σ. The product of two permutations π, σ is defined by $(\pi\sigma)(i) = \pi(\sigma(i))$, for all $i \in [n]$. The collection of all permutations on $[n]$, denoted by $\mathbb{S}_n$, is a group with the permutation product as the group operation.

A cycle of length k, denoted by $(i_1 \cdots i_k)$, is a permutation that acts on $[n]$ by cyclically shifting the order of elements in $\{i_1, \cdots, i_k\}$, and keeping all other elements fixed. The set $\{i_1, \cdots, i_k\}$ is the *support* of the cycle, denoted by $supp((i_1 \cdots i_k))$. A cycle of length two is called a *transposition*, and for $i_1 = a$, $i_2 = b$ it is denoted by $\tau = (a\, b)$. An *adjacent transposition* is the transposition of two adjacent elements in the order. The symbol e is reserved for the identity permutation $(1, 2, \cdots, n)$.

Two cycles are said to be *disjoint* if the intersection of their supports is empty; on the other hand, *adjacent cycles* share *exactly one* element. It is well known that a permutation can be written as a product of disjoint cycles, which is often referred to as the *cycle decomposition*. As will be shown, a cycle can also be decomposed into a product of shorter cycles representing a combination of disjoint and adjacent cycles. This procedure is termed an *adjacent cycle decomposition* and was not previously studied in sorting problems.

B. Problem Formulation

If $\sigma = \pi\tau_1 \cdots \tau_k$, where τ_i is a transposition for all $1 \le i \le k$, we refer to $\tau_1 \cdots \tau_k$ as a transform converting π into σ. The set of all such transforms is denoted by $A(\pi, \sigma)$. If the transpositions τ_i are restricted to be adjacent, we denote the corresponding set of transforms by $A_a(\pi, \sigma)$. The Cayley and Kendall distances equal the smallest number of transpositions and adjacent transpositions in any transform of $A(\pi, \sigma)$ and $A_a(\pi, \sigma)$, respectively. These distances may be generalized as follows. Given a weight function $\boldsymbol{\varphi}$ that assigns to each transposition τ a positive weight φ_τ, the $\boldsymbol{\varphi}$-weighted transposition distance between π and σ is defined by

$$d_\varphi(\pi, \sigma) = \min_{\tau_1 \cdots \tau_k \in A(\pi, \sigma)} \Sigma_{i=1}^{k} \varphi_{\tau_i}. \tag{1}$$

Let $G_\varphi = (V, E)$, with $V = [n]$, be a connected, undirected, edge-weighted graph, with all weights positive. Assume that $\varphi_{(a\, b)}$ equals the minimum weight among all paths between vertices a and b, for all $a, b \in [n]$. Then $\boldsymbol{\varphi}$ is a graph metric, and we refer to $G_\varphi = (V, E)$ as the *defining graph* of the weight function $\boldsymbol{\varphi}$. An arbitrary chosen minimum weight path between vertices a and b is denoted by $p^*_\varphi(a, b)$. We henceforth focus on the family of graph metric weights, satisfying the triangle inequality. In particular, a weight function $\boldsymbol{\varphi}$ is a *metric-tree* weight function if it has a tree-structured defining graph. For such defining graphs, there exists a unique minimum cost path between any two vertices. Furthermore, 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_φ. If G_φ is a path, then $\boldsymbol{\varphi}$ is called a *metric-path* weight function. If there exists a unique vertex in a tree structured G_φ of highest degree equal to three, the graph is called a *Y-tree*. The vertex with highest degree is referred to as the *central vertex*.

In order to characterize $d_\varphi(\pi, \sigma)$, we use of the *displacement* of π and σ, denoted by $D_\varphi(\pi, \sigma)$, and defined as

$$D_\varphi(\pi, \sigma) = \sum_{i=1}^{n} \mathrm{wt}(p^*_\varphi(\pi^{-1}(i), \sigma^{-1}(i))). \tag{2}$$

Here, $\mathrm{wt}(\cdot)$ stands for the weight of the path and $\mathrm{wt}(p^*_\varphi(\pi^{-1}(i), \sigma^{-1}(i)))$ is referred to as the *displacement* of the element i in the permutations π, σ. It is easy to verify that for every positive weight function, the weighted transposition distance d_φ and the displacement D_φ are both pseudo-metrics and left-invariant (i.e., $d_\varphi(\pi, \sigma) = d_\varphi(\omega\,\pi, \omega\,\sigma)$ and $D_\varphi(\pi, \sigma) = D_\varphi(\omega\,\pi, \omega\,\sigma)$, for all $\pi, \sigma, \omega \in \mathbb{S}_n$). We therefore concentrate on studying $d_\varphi(\sigma, e)$ and $D_\varphi(\sigma, e)$, and refer to the underlying problem as the (weighted) *decomposition* problem.

III. Similarity Distances for Cycle Permutations

To devise an algorithm for computing the similarity distance between a permutation and the identity under Y-tree weight functions, we first focus on the case of a single cycle. Denote the branches of the defining Y-tree, representing paths starting from the central vertex and extending to a leaf (and excluding the central vertex), as (B_1, B_2, B_3). The underlying cycle is naturally represented as a collection of directed edges indicating the order of the elements in the cycle, and the edges may cross branches or bridge elements on the same branch.

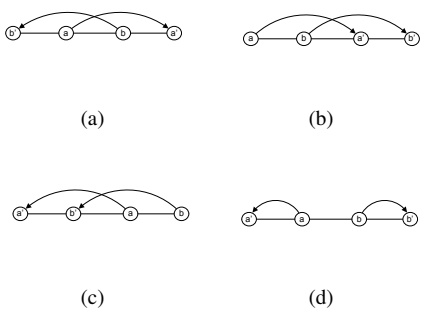

Fig. 2: Four types of local structures used to assess the efficiency of a transposition, with $a' = \sigma(a)$ and $b' = \sigma(b)$.

A cycle path on the Y-tree is a cycle that has support contained in $B_i \cup B_j$, for some i, j not necessarily distinct. In other words, the cycle lies on a path if its support is contained in the support of at most two of the three branches. Furthermore, for a branch pair (B_i, B_j), $i \neq j$, let l_{ij} be the number of directed edges from $a_i \in B_i$ to $a_j \in B_j$. If for a cycle permutation $l_{ij} = l_{ji}$, we say that the branch pair (B_i, B_j) of the cycle is balanced. Furthermore, if $l_{ij} = l_{ji}$ for all $i, j \in \{1, 2, 3\}$, we say that the cycle is *balanced*.

Definition 1. *A transposition $(a\,b)$ is efficient with respect to the permutation σ if*

$$D(\sigma, e) - D\left(\sigma(a\,b), e\right) = 2\varphi_{(a\,b)}. \tag{3}$$

The inefficiency $\varpi_{(a\,b)}$ of a transposition $(a\,b)$ with respect to the permutation σ is non-negative, and defined as

$$\varpi_{(a\,b)} = 2\varphi_{(a\,b)} - (D(\sigma, e) - D(\sigma(a\,b), e)). \tag{4}$$

Intuitively, one can view efficient transpositions as transpositions that reduce the transposition displacement weight in an optimal fashion. Hence, our analysis starts by providing a characterization of efficient transpositions. For this purpose, observe that any two vertices a, b in the support of a cycle embedded in a Y-tree may assume only one of the four possible configurations shown in Fig. 2. The unique path between the two vertices a, b of G_φ is denoted by a-b.

Lemma 1. *Transposition $(a\,b)$ is efficient if and only if vertex a lies on the b-$\sigma(b)$ path in G_φ and vertex b lies on the a-$\sigma(a)$ path in G_φ (hence, only case (a) of Fig. 2 corresponds to an efficient transposition).*

Proof: There exists a unique a-$\sigma(a)$ path, and thus

$$\begin{aligned}&\mathrm{wt}(p^*_\varphi(\sigma^{-1}(a), e(a))) = \mathrm{wt}(p^*_\varphi(a, \sigma(a)))\\ &= \mathrm{wt}(p^*_\varphi(a, b)) + \mathrm{wt}(p^*_\varphi(b, \sigma(a))) = \varphi_{(a\,b)} + \mathrm{wt}(p^*_\varphi(b, \sigma(b))),\end{aligned}$$

where the first equality follows from left-invariance of the distance, the second equality holds due to the fact that b lies on the a-$\sigma(a)$ path, while the third equality is a consequence of the definition of metric-tree weight functions. A similar expression holds for b. Upon applying the transposition $(a\,b)$, the displacement of a decreases by $\varphi_{(a\,b)}$, while the displacement of b decreases by $\varphi_{(a\,b)}$. As the transposition $(a\,b)$ can only affect the displacement of vertices a and b, the total reduction of displacement equals $2\varphi_{(a\,b)}$.

Without loss of generality, suppose that $(a\,b)$ is efficient but a does not lie on the b-$\sigma(b)$ path. Then, the net displacement of b reduced by $(a\,b)$ is strictly less than $\varphi_{(a\,b)}$, implying that the total reduction in displacement is strictly less than $2\varphi_{(a\,b)}$. This contradicts the assumption that $(a\,b)$ is efficient. ■

Theorem 1. *The distance between σ and e is bounded as*

$$d_\varphi(\sigma, e) \geq \frac{1}{2} D_\varphi(\sigma, e). \tag{5}$$

The lower bound is achieved for metric-path weight functions.

The proof relies on a technique described by the authors in [5], which is detailed in the full version of the paper [11].

We describe next an algorithm for computing a minimum cost transform $\tau_1 \cdots \tau_k$ for a metric-path cost. Label the vertices in the defining path from left to right as $1, 2, \cdots, n$. Suppose that the given cycle is $\sigma = (a_1\, a_2\, \ldots\, a_{|supp(\sigma)|})$, where $a_1 = \min\ supp(\sigma)$. If this is not the case, simply rewrite σ by cyclically shifting its elements. Let $a_t = \min_{i \in supp(\sigma)}\{i : i \neq a_1\}$ be the closest element to a_1 in $\mathcal{G}(\sigma)$. The decomposition procedure is given in **Algorithm 1**.

Algorithm 1: Path Case Decomposition

Input: A cycle $\sigma = (a_1\, a_2\, \ldots\, a_{|supp(\sigma)|})$
Output: A minimum cost decomposition $\sigma = \tau_1 \tau_2 \cdots \tau_k$

```
while |supp(σ)| > 2 do
    if a_t ≠ a_{|supp(σ)|} then
        σ = σ1σ2, where σ1 = (a1 a2 ··· at) and
        σ2 = (at a_{t+1} ··· a_{|supp(σ)|});
        repeat procedure for σ1 and σ2;
    else
        σ = σ1τ, where τ = (a1 at) and
        σ1 = (a2 a3 ··· a_{|supp(σ)|});
        repeat procedure for σ1.
    end
end
```

By following through the steps of **Algorithm 1** and by recalling the properties of efficient transpositions from Lemma 1, it can be shown that cycles with path costs can always be decomposed with the minimum weight guaranteed by Theorem 1, since all transpositions used *are efficient*.

Fig. 3 illustrates why the path-cost decomposition algorithm cannot be generalized to Y-tree weight functions: Here, the total displacement $D_\varphi((1\,2\,3), e)$ equals 6, while

$$d_\varphi((1\,2\,3), e) = 4 \neq \frac{1}{2} D_\varphi((1\,2\,3), e).$$

Note that in Fig. 3, the central vertex does not belong to the support of the cycle, and furthermore, the cycle is not balanced. It is not clear if this and other cycles may be decomposed using efficient transpositions only, and if so,

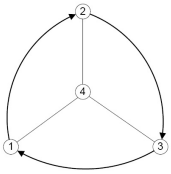

Fig. 3: A weight φ defined via a Y-tree with all edges of weight one; the cycle equals $(1\,2\,3)$. Thin lines represent the Y-tree $\mathcal{G}$, while directed bold lines represent the cycle.

how to find such a decomposition. To address this question, we consider three different scenarios: 1) the central vertex belongs to the support of the cycle; 2) the central vertex does not belong to the support of the cycle, but the cycle is balanced; 3) neither of the two aforementioned conditions hold. Decomposition algorithms for case 1) and 2) comprise two subroutines: One, for rewriting the cycle into a product of shorter adjacent cycles, each of which has the property that its support lies on a *path* in the Y-tree. And another subroutine, which decomposes cycles that have supports on paths using efficient transpositions. Case 3) calls for additional processing steps.

A. Case 1) and 2)

Denote the central vertex by a_0. The first subroutine for cycles that contain the central vertex is described in **Algorithm 2**. We use k to track the index of subcycles in the decomposition generated at each iteration. The indicator functions $\mathbb{I}$ use conditions of the form $supp(\sigma) \cap B_i \neq \emptyset$, where $i \in \{1,2,3\}$. Note that a_1 may correspond to different vertices from iteration to iteration. The algorithm terminates when all subcycles σ_j have supports contained in a path.

Algorithm 2: Adjacent Cycle Decomposition 1

Input: A cycle $\sigma = (a_0\, a_1\, \cdots\, a_{|supp(\sigma)|-1})$
Output: An adjacent cycle decomposition $\sigma = \sigma_{k+1}\cdots\sigma_1$ with supports of σ_j, $j = 1,\, 2\, \cdots, k$, lying on paths of the Y-tree

Initialize: $k \leftarrow 0$;
while $\mathbb{I}_{\{supp(\sigma)\cap B_1\neq\emptyset\}} \cdot \mathbb{I}_{\{supp(\sigma)\cap B_2\neq\emptyset\}} \cdot \mathbb{I}_{\{supp(\sigma)\cap B_3\neq\emptyset\}} = 1$ **do**
  $k \leftarrow k+1$;
  Assuming that $a_1 \in B_1$, let $t = \min_{i\in\{1,\cdots,|supp(\sigma)|-1\}} \{i : a_i \in B_1 \text{ and } a_{i+1} \notin B_1\}$;
  $\sigma_k \leftarrow (a_0\, a_1\, \cdots\, a_t)$ and $\sigma' \leftarrow (a_0\, a_{t+1}\, \cdots\, a_{|supp(\sigma)|-1})$;
  $\sigma \leftarrow \sigma'$;
end
$\sigma_{k+1} \leftarrow \sigma$.

Theorem 2. *For a cycle σ containing the central vertex, the distance between σ and e equals*

$$d_\varphi(\sigma, e) = \frac{1}{2} D_\varphi(\sigma, e).$$

Proof: It suffices to prove that $d_\varphi(\sigma,e) \le \frac{1}{2} D_\varphi(\sigma,e)$. Let $\sigma = \sigma_{k+1}\cdots\sigma_1$ be the cycle decomposition of σ generated by **Algorithm 2**. Then,

$$d_\varphi(\sigma, e) \le \sum_{i=1}^{k+1} d_\varphi(\sigma_i, e) = \frac{1}{2}\sum_{i=1}^{k+1} D_\varphi(\sigma_i, e), \tag{6}$$

which follows from the triangle inequality and the fact that all adjacent cycles have supports on paths. From Theorem 1, we know that for metric paths, one has $d_\varphi(\sigma_i,e) = \frac{1}{2} D_\varphi(\sigma_i,e)$. To complete the proof, we need to show

$$D_\varphi(\sigma, e) = \sum_{i=1}^{k+1} D_\varphi(\sigma_i, e). \tag{7}$$

In **Algorithm 2**, σ is updated at each round. For ease of exposition, denote the decomposition of σ in round j by $\sigma^{(j)}$, for $j = 0, 1, \cdots, k$. At each iteration of **Algorithm 2**,

$$\sigma^{(j)} = \sigma_{j+1}\sigma^{(j+1)}.$$

To show that (7) holds, we need to demonstrate that

$$D_\varphi(\sigma^{(j)}, e) = D_\varphi(\sigma_{j+1}, e) + D_\varphi(\sigma^{(j+1)}, e) \tag{8}$$

for all $j = 0, 1, \cdots, k$. Since $\sigma_{j+1} = (a_0^{(j)}\, a_1^{(j)}\, \cdots\, a_t^{(j)})$ and $\sigma^{(j+1)} = (a_0^{(j)}\, a_{t+1}^{(j)}\, \cdots\, a_{|supp(\sigma^{(j)})|-1}^{(j)})$, the right hand side of (8) equals

$$\begin{aligned}
&\sum_{i=0}^{t-1} \mathrm{wt}(p_\varphi^*(a_i^{(j)}, a_{i+1}^{(j)})) + \sum_{i=t+1}^{|supp(\sigma^{(j)})|-2} \mathrm{wt}(p_\varphi^*(a_i^{(j)}, a_{i+1}^{(j)})) \\
&+ \mathrm{wt}(p_\varphi^*(a_t^{(j)}, a_0^{(j)})) + \mathrm{wt}(p_\varphi^*(a_0^{(j)}, a_{t+1}^{(j)})) \\
&+ \mathrm{wt}(p_\varphi^*(a_{|supp(\sigma^{(j)})|-1}^{(j)}, a_0^{(j)})) = \mathrm{wt}(p_\varphi^*(a_{|supp(\sigma^{(j)})|-1}^{(j)}, a_0^{(j)})) \\
&+ \sum_{i=0}^{|supp(\sigma^{(j)})|-2} \mathrm{wt}(p_\varphi^*(a_i^{(j)}, a_{i+1}^{(j)})) = D_\varphi(\sigma^{(j)}, e),
\end{aligned}$$

as desired. As the parameter t is chosen so that $a_t^{(j)} \in B_1$ and $a_{t+1}^{(j)} \notin B_1$, we have

$$\mathrm{wt}(p_\varphi^*(a_t^{(j)}, a_{t+1}^{(j)})) = \mathrm{wt}(p_\varphi^*(a_t^{(j)}, a_0^{(j)})) + \mathrm{wt}(p_\varphi^*(a_0^{(j)}, a_{t+1}^{(j)})).$$

This completes the proof. ∎

Note that in **Algorithm 2**, we started with the directed edge emanating from the central vertex and checked if the decomposition conditions are met. When $a_0 \notin supp(\sigma)$, it is not clear which edge to start with in order create a proper adjacent decomposition. Hence, a specialized search procedure is needed to identify good starting edge(s), described in the full version of the paper [11]. Using proof techniques similar to those previously described, we arrive at the following result.

Theorem 3. *For a balanced cycle σ, the distance between σ and e equals*

$$d_\varphi(\sigma, e) = \frac{1}{2} D_\varphi(\sigma, e).$$

B. Case 3: Unbalanced Cycles

We first show that if for some $i \neq j \in \{1, 2, 3\}$, one has $l_{ij} \neq l_{ji}$, equality in Theorem 1 cannot be achieved.

Theorem 4. *For any unbalanced cycle σ we have*

$$d_\varphi(\sigma, e) > \frac{1}{2} D_\varphi(\sigma, e),$$

or more precisely,

$$d_\varphi(\sigma, e) = \frac{1}{2} D_\varphi(\sigma, e) + \min_{a_i \in supp(\sigma)} \varphi_{(a_0\, a_i)}.$$

The proof of the theorem is highly technical and therefore relegated to the full version of the paper [11]. The unbalanced cycle decomposition procedure is given in **Algorithm 3**.

Algorithm 3: Adjacent Cycle Decomposition 2

Input: A cycle $\sigma = (a_1\, a_2\, \cdots\, a_{|supp(\sigma)|})$
Output: An adjacent cycle decomposition
$\sigma_{k+1} \cdots \sigma_1 (a_0\, a_j)(a_j\, a_{j+1})$, a_j and a_{j+1} on the same branch;
$\sigma_{k+1} \cdots \sigma_1 (a_0\, a_j)$, a_j and a_{j+1} on different branches,
with the supports of σ_i, $i = 1, \cdots, k+1$, lying on paths of the Y-tree

1 $a_j \leftarrow \text{argmin}_{a_i \in supp(\sigma)} \varphi_{(a_0\, a_i)}$;
2 **if** a_j *and* a_{j+1} *lie on the same branch* **then**
3 $\quad \sigma^* \leftarrow \sigma(a_j\, a_{j+1})$;
4 **else**
5 $\quad \sigma^* \leftarrow \sigma$;
6 **end**
7 $\sigma^* \leftarrow \sigma^*(a_0\, a_j)$;
8 Call **Algorithm 2** to obtain $\sigma^* = \sigma_{k+1} \cdots \sigma_1$.

IV. General Permutations

For general permutations, we describe next a linear time $5/3$-approximation algorithm, **Algorithm 4**. The only step that introduces inefficiency in the decomposition process is merging the central vertex into the cycle, as performed in step 6 of **Algorithm 4**. As a result, $d(\sigma, e)$ may be bounded as

$$\frac{1}{2} D(\sigma, e) \leq d(\sigma, e) \leq \frac{1}{2} D_\varphi(\sigma, e) + \varphi_{(a_0\, a_{k_1})}.$$

It can also be shown that $\varphi_{(a_0\, a_{k_1})} \leq \frac{1}{3} D(\sigma, e)$, which leads to the claimed approximation factor of $5/3$. Detailed derivations may be found in the full version of the paper [11]. Note that provable lower bounds on the approximation algorithm are currently unknown.

Remark. It is easy to see that most routines of our Y-tree sorting algorithms are parallelizable. Consequently, two questions of interest are to determine if the Y-tree decomposition problem is *3SUM-hard* (i.e., if solving the problem in subquadratic time implies a subquadratic algorithm for the 3SUM problem) and if the decision problem is NC (i.e., in Nick's class, and consequently decidable in polylog time on a parallel computer). These questions will be addressed elsewhere.

Acknowledgment. This work was supported in part by NSF grants CCF 0809895, CCF 1218764 and the Emerging Frontiers for Science of Information Center, CCF 0939370.

Algorithm 4: Constant-Approximation Algorithm for General Permutations

Input: An arbitrary permutation σ
Output: A constant-approximation transform of σ

1 Decompose all the cycles with supports on paths and cycles containing the central vertex independently; denote the result by σ^*;
2 **if** $\sigma^* = e$ **then**
3 $\quad$ Stop;
4 **end**
5 Let $a_{k_1}, a_{k_2}, a_{k_3}$ be farthest nonfixed points with outgoing edges across branches on B_1, B_2, B_3, respectively; in addition, assume $\varphi_{(a_0\, a_{k_1})} = \min\{\varphi_{(a_0\, a_{k_1})}, \varphi_{(a_0\, a_{k_2})}, \varphi_{(a_0\, a_{k_3})}\}$;
6 $\sigma^* \leftarrow \sigma^*(a_0\, a_{k_1})$;
7 Invert σ^* to obtain $(\sigma^*)^{-1}$;
8 Given m cycles in $(\sigma^*)^{-1}$ with edges across branches, let $b_1, b_2, \cdots, b_m \in B_1$ be the heads of the outgoing edges, with labels indicating their proximity to the central vertex, a_0. Let $b_0 = a_0$;
9 **for** *i=1:m* **do**
10 $\quad \sigma^* \leftarrow \sigma^*(b_i\, b_{i-1})$;
11 **end**
12 Call **Algorithm 1 and 2**.

References

[1] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J.ACM*, 55(5):23:1–23:27, 2008.

[2] Stanislav Angelov, Keshav Kunal, and Andrew McGregor. Sorting and selection with random costs. In *LATIN 2008: Theoretical Informatics*, pages 48–59. Springer, 2008.

[3] Alexander Barg and Arya Mazumdar. Codes in permutations and error correction for rank modulation. *Information Theory, IEEE Transactions on*, 56(7):3158–3165, 2010.

[4] Persi Diaconis and R. L. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(2):262–268, 1977.

[5] Farzad Farnoud (Hassanzadeh) and Olgica Milenkovic. Sorting of permutations by cost-constrained transpositions. *IEEE Trans. Inf. Theor.*, 58(1):3–23, 2012.

[6] Anxiao Jiang, Robert Mateescu, Moshe Schwartz, and Jehoshua Bruck. Rank modulation for flash memories. *Information Theory, IEEE Transactions on*, 55(6):2659–2673, 2009.

[7] Oren Kapah, GadM. Landau, Avivit Levy, and Nitsan Oz. Interchange rearrangement: The element-cost model. In Amihood Amir, Andrew Turpin, and Alistair Moffat, editors, *String Processing and Information Retrieval*, volume 5280 of *Lecture Notes in Computer Science*, pages 224–235. Springer Berlin Heidelberg, 2009.

[8] Ravi Kumar and Sergei Vassilvitskii. Generalized distances between rankings. In *Proceedings of the 19th international conference on World wide web*. ACM, 2010.

[9] M. Elena Renda and Umberto Straccia. Web metasearch: rank vs. score based rank aggregation methods. In *Proceedings of the 2003 ACM symposium on Applied computing*. ACM, 2003.

[10] D Sculley. Rank aggregation for similar items. In *In SDM. SIAM*, pages 587–592. Citeseer, 2007.

[11] Lili Su, Farzad Farnoud, and Olgica Milenkovic. Computing similarity distances between rankings. *CoRR*, abs/1307.4339, 2013.