Sorting of Permutations by Cost-Constrained Transpositions

Farzad Farnoud (Hassanzadeh), *Student Member, IEEE*, and Olgica Milenkovic, *Member, IEEE*

***Abstract*—The problem of finding a minimum decomposition of a permutation in terms of transpositions with predetermined non-uniform and non-negative costs is addressed. Alternatively, computing the transposition distance between two permutations, where transpositions are endowed with arbitrary non-negative costs, is studied. For such cost functions, polynomial-time, constant-approximation decomposition algorithms are described. For metric-path costs, exact polynomial-time decomposition algorithms are presented. The algorithms in this paper represent a combination of Viterbi-type algorithms and graph-search techniques for minimizing the cost of individual transpositions, and dynamic programing algorithms for finding minimum cost decompositions of cycles. The presented algorithms have a myriad of applications in information theory, bioinformatics, and algebra.**

***Index Terms*—Cost function, decomposition, distance, permutation, sorting, transposition.**

I. Introduction

PERMUTATIONS are ubiquitous combinatorial objects encountered in areas as diverse as mathematics, computer science, communication theory, and bioinformatics. The set of all permutations of n elements—the symmetric group of order n, $\mathbb{S}_n$—plays an important role in algebra, representation theory, and analysis of algorithms [3]–[6]. As a consequence, the properties of permutations and the symmetric group have been studied extensively.

One of the simplest ways to generate an arbitrary permutation is to apply a sequence of transpositions—swaps of two elements—to a given permutation, usually the identity permutation. The sequence of swaps can be reversed in order to recover the identity permutation from the original permutation. This process is referred to as sorting by transpositions.

A simple result, established by Cayley in the 1860's, asserts that the minimum number of transpositions needed to sort a permutation so as to obtain the identity permutation is the difference of the size of the permutation and the number of cycles formed by the elements of the permutation. Cayley's result is based on a simple constructive argument, which reduces to a linear-complexity procedure for breaking cycles into sub-cycles. The number of transpositions used in sorting a permutation is equivalent to the transposition distance between the permutation and the identity permutation. Since permutations form a group, the transposition distance between two arbitrary permutations equals the transposition distance between the identity permutation and the composition of the inverse of one permutation and the other permutation.

We address a substantially more challenging question: assuming that each transposition has a non-negative, but otherwise arbitrary cost, is it possible to find the minimum sorting cost and the sequence of transpositions used for this sorting in polynomial time? In other words, can one compute the cost-constrained transposition distance between two permutations in polynomial time? Although at this point it is not known if the problem is NP hard, at first glance, it appears to be computationally difficult, due to the fact that it is related to finding minimum generators of groups and the subset-sum problem [7]. Nevertheless, we show that large families of cost functions—such as costs based on metric-paths—have exact polynomial-time decomposition algorithms. Furthermore, we devise algorithms for approximating the minimum sorting cost for any non-negative cost function, with an approximation constant that does not exceed four. The presented algorithms represent an amalgamation of well known algorithms from information and coding theory, such as the Viterbi algorithm and dynamic programming methods. Furthermore, our proof techniques are based on a number of graph theoretic methods frequently used in coding and information theory.

Our investigation is motivated by four different applications. The first application pertains to sorting of genomic sequences [8]–[11], while the second application is related to a generalization of the notion of a chemical channel (also known as trapdoor channel [12]–[14]). The third application is in the area of coding for storage devices [15]–[18], while the fourth arises in the area of network security [19]–[21]. These problems are discussed in detail in the Motivation section.

The paper is organized as follows. Section II describes a class of motivating applications, while Section III introduces the notation followed in the remainder of the paper, as well as relevant definitions. Sections IV, V, and VI contain the main results of our study: a three-stage polynomial-time approximation algorithm for general cost-constrained sorting of permutations, an exact polynomial-time algorithm for sorting with metric-path

Manuscript received July 17, 2010; revised June 17, 2011; accepted July 14, 2011. Date of current version January 06, 2012. This work was funded by the National Science Foundation (NSF) under Grant NSF CCF 08-21910, Grant NSF CCF 08-09895, and Grant NSF CCF 0644427. The material in this paper was presented in part at the 2010 IEEE Information Theory Workshop, Dublin, Ireland and in part at the 2011 IEEE International Symposium on Information Theory.

F. Farnoud (Hassanzadeh) is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Champaign, IL 61820 USA (e-mail: hassanz1@uiuc.edu).

O. Milenkovic is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: milenkov@uiuc.edu).

Communicated by E. Viterbo, Associate Editor for Coding Techniques.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2011.2171532

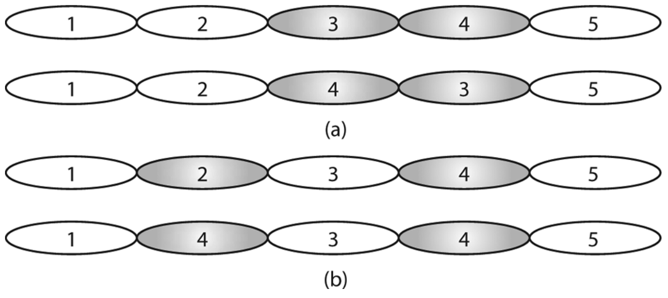

Fig. 1. Transposition of adjacent blocks (a) and non-adjacent blocks (b) in a DNA sequence.

costs, as well as a complexity analysis of the described techniques. Section VII contains the concluding remarks.

II. Motivation

A. Genomic Rearrangement

Genomic strings—such as DNA sequences—evolve from common ancestors, and therefore frequently contain conserved substrings that encode proteins vital for the survival of all organisms. Although these substrings have an almost identical composition, in different species they appear at disparate locations. This may be attributed to the fact that genomic strings tend to break, either due to external factors such as radiation or due to internal processes that adversely affect the integrity of the sequences. Once a string breaks off from its original location, it may cause cellular death if the breakage occurred within a coding region, or it may leave the organism intact, provided that breakage did not disrupt the functionality of the string. Upon breaking off, the functional unit may reattach or insert itself at some other location in the genome. This breakage phenomenon is believed to be the explanation for the observed evolutionary reshuffling of functional substrings in genomes.

Permutations of functional substrings also occur during tumorogenesis and are often considered major markers of the disease [8]. Genomes of cancer cells tend to contain the same sequence of blocks as normal cells, but erroneously redistributed among the chromosomes in a cell. These mutational events that characterize cancer are known as translocations (when substrings of two different chromosomes are exchanged), transpositions (when substrings—not necessarily of the same length—of the same chromosome are exchanged)[1], or reversals (when symbols of a single substring are reversed in order). Figs. 1 and 2 illustrate these concepts pictorially. Notice that translocations and transpositions require four breakpoints, while adjacent transpositions require three breakpoints. Reversals occur in the presence of only two breakpoint. It is widely believed that the higher the number of breakpoints needed for the mutation to occur, the less likely the mutation tends to be.

Recently, it was observed that breakage does not occur randomly throughout the genome—there exist certain sites in the sequence that are much more prone to breakage than others.

[1]Notice the different usage of the term "transposition" in the bioinformatics literature. There, a transposition describes an exchange in order of two substrings and not two individual symbols. [22]

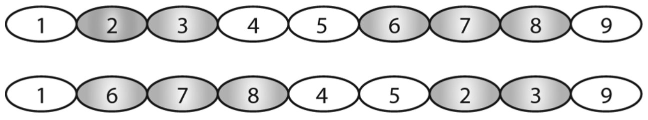

Fig. 2. Transposition of blocks of different length in a DNA sequence.

These so called "fault lines" or "fragile regions" are very frequently related to the composition of the sequence—for example, regions rich in A and T nucleotides that form hairpin or crucifix-shaped protrusions in the DNA sequence [9]–[11].

This finding motivated a large body of work on developing efficient algorithms for reverse-engineering the sequence of shuffling steps performed on conserved subsequences. With a few exceptions, most of the methods for sorting use reversals rather than transpositions, they follow the uniform cost model (each change in the ordering of the blocks is equally likely) and the most parsimonious sorting scenario (the sorting scenario with smallest number of changes is the most likely explanation for the observed order). Several approaches that do not fit into this framework were described in [23], [24]. Sorting by cost-constrained transpositions can be seen as a special instance of the general subsequence sorting problem, where the sequence is allowed to break at three or four points. Furthermore, it allows for accommodating the sequence composition information via the cost function. Although an exact and mathematically precise determination of breakage probability and consequently transposition cost is highly unlikely to be conducted, one can start with simple models were the cost increases inversely proportional to the AT content of the borders of the subsequences. Other, more precise models are possible as well but they fall out of the scope of this paper. Note that reverse engineering the sequence of transpositions and breakages under this model may provide valuable information about the pathway of tumorogenesis and shed new insight into the genetic causes of chromosome rearrangement in cancer.

B. The Trapdoor Channel

The second application arises in the study of so called chemical channels. The chemical channel is a channel model introduced in information theory, motivated by a simple view of a causal information processing in cells. In this model, binary symbols are used to describe molecules, as it is assumed that only two types of molecules are possible [12]–[14]. The channel is initialized with a primer molecule, and afterwards, molecules from a queue are sequentially pushed into the channel (well). At each step of the procedure, one of the two molecules in the well is randomly selected as channel output. This processing system has unit memory, which results in allowing only transpositions of adjacent elements in the permutation (which by the duality of position/symbol value described in Section III amounts to adjacent symbol transpositions). The question at hand is how can one find the original channel input provided that the channel output is known? Alternatively, this question reduces to one of decoding the output of the trapdoor channel. Computing the adjacent transposition distance between the input and output sequences represents a crucial step in this decoding process.

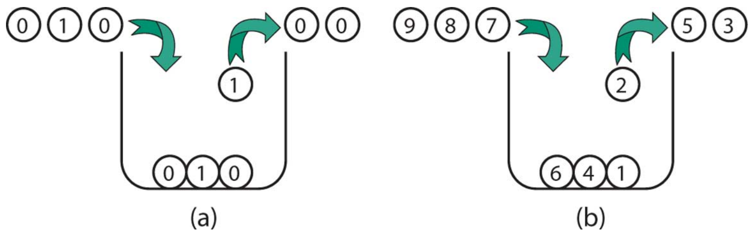

Fig. 3. The binary trapdoor channel (a) and a generalization of the trapdoor channel (b) both with memory larger than one.

This model has several drawbacks in terms of representing potential biological channels. First, in the context of contextual and causal cellular functioning, all molecules should be viewed as distinguishable. Consequently, all symbols in the sequence should be different and the channel input should be representable by a permutation. An illustration can be seen in Fig. 3(b). Furthermore, for some functions, more than two molecules should be allowed to interact simultaneously, leading to the requirement that the channel well may hold more than two symbols (i.e., have memory larger than one). Additionally, since every chemical reaction has to be timed, the channel model should allow for the option that at some time instant no molecules is pushed out of the channel or that some molecules have higher probabilities than others of being sent out of the channel. These extended assumptions allow one to view this generalization of the trapdoor channel as a channel that transposes elements in the output permutation according to a non-uniform cost model. Note that in this case, although all transpositions are possible, not all output permutations are plausible for one given input, since the transpositions used have costs that depend on the input permutation and are updated sequentially as molecules are being sent out of the well.

C. Rank Modulation Coding

The third application is concerned with flash memories and rank permutation coding (see [15] and [16]). In this case, one is given an array of cells that may store charge levels. Due to aging, environmental conditions and design, the cells are subjected to charge leakage, leading to the phenomena of low memory endurance [18]. Due to this leakage, the information content of the cells may be compromised beyond the possibility of correction.

One way to mitigate this problem is to use the idea of rank modulation, which is based on the framework of storing information through the ordering of cell charges rather than the charge levels themselves. In this scenario, a storage error occurs only if a charge of higher level leaks below the level of a previously less charged cell. An example is shown in Fig. 4. Most leakage models assume nearly uniform relative cell charge changes, allowing only for the possibility of exchanging the ranking of two consecutive symbols. In this case, the distance metric for input-output sequences of flash memories is the so-called Kendall distance between permutations [17]. Kendall's distance measures the smallest number of (pairwise) adjacent transpositions needed to transform one permutation into another.

Recent experiments reveal that cell leakage levels depend both on the location of the cells in the array as well as their initial charge level [18]. If one considers more precise charge leakage

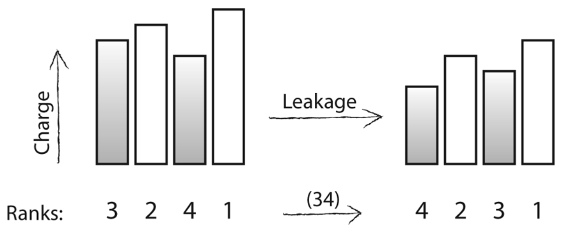

Fig. 4. On the left-hand side the initial state of the storage is shown. The electric charge in each cell determines its rank. Because of different leakage rates in the shaded cells, their rank is swapped, as seen on the right-hand side.

models for memory cells, the costs of adjacent transpositions become non-uniform and non-adjacent transpositions may appear with non-zero probability as well. The first case can easily be captured by a transposition cost model in which non-adjacent transpositions have unbounded cost, while the costs of adjacent transpositions are unrestricted. Hence, the proposed decomposition algorithms can be used as part of general soft-information rank modulation decoders.

D. Packet Reordering in Networks

Packet reordering is a high-incidence event in networks, and in particular, in the Internet. Usually, it can be attributed to multi-path routing where different paths experience different delays. Due to the fact that packet reordering can compromise the performance of the otherwise fairly robust TPC protocol, several network attack strategies were developed for the purpose of delaying efficient service. The basis of these attacks is that packet reordering causes unnecessary retransmission. TCP operates in the following manner: when receiving a packet out of order, the protocol sends several ACK signals to initiate fast packet retransmission since the packet may have been lost. If the change in order is not due to packet loss, but rather caused by reordering, then the retransmissions are unnecessary and may cause congestion. Furthermore, frequent retransmission requests may prompt senders to see them as indicators of network congestion, and this in turn triggers reduction in the transmission rate. Finally, the delay of the network receivers is significantly increased due to re-orderings, since the message content has to be passed on to higher network layers in order, which introduces the need for packet buffering.

The question of interest in this case is to reverse engineer the attacker's strategy. Due to the fact that most attackers will not have access to most of the nodes in the network, and due to the partly random process of packet routing, only certain packet reorderings will be plausible or likely. The underlying transpositions will consequently have nonuniform cost, although there

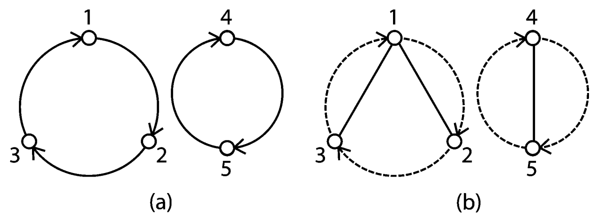

Fig. 5. (a) The digraph $\mathcal{G}(\pi)$ of the permutation $\pi = (123)(45)$. (b) The graph $\mathcal{T}(\tau)$, for the decomposition $\tau = (13)(12)(45)$ of π. Edges of $\mathcal{G}(\pi)$ and $\mathcal{T}(\tau)$ are represented with dashed and solid lines, respectively.

remains the difficult problem of estimating or approximating the costs of transpositions. One reasonable assumption for the attacker model is that the compromised nodes may be in close proximity in the network, and that the cost will depend on the distance of the compromised nodes from some "central node" or "central cluster". This information, of course, can only be obtained through thorough network traffic analysis as performed, for example, in [19], [20].

For a different study of packet reordering in networks the interested reader is referred to [21].

III. Notation and Definitions

A permutation π of $[n] := \{1, 2, \dots, n\}$ is a bijection from $[n]$ to itself. The set of permutations of $[n]$ is denoted by $\mathbb{S}_n$. A permutation can be represented in several ways. In the two-line notation, the domain is written on top, and its image below. The one-line representation is the second row of the two-line representation. A permutation may also be represented as the set of elements and their images.

For example, one can write a permutation π as $\pi(1) = 3$, $\pi(2) = 1$, $\pi(3) = 2$, $\pi(4) = 5$, $\pi(5) = 4$, or more succinctly as $\pi = 31254$, or in the two-line notation as

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

The product $\pi_2\pi_1$ of two permutations π_1 and π_2 is the permutation obtained by first applying π_1 and then π_2, i.e., the product represents the composition of π_1 and π_2.

The *functional digraph* of a function $f : [n] \to [n]$, denoted by $\mathcal{G}(f)$, is a directed graph with vertex set $[n]$ and an edge from i to $f(i)$ for each $i \in [n]$. We use the words vertex and element interchangeably.

For a permutation π of $[n]$, $\mathcal{G}(\pi)$ is a collection of disjoint *cycles* since the in-degree and out-degree of each vertex is exactly one. The cycles of a permutations are the cycles of its functional digraph. Each cycle can be written as a k—tuple $(a_1 a_2 \cdots a_k)$, where k is the length of the cycle and there is an edge from a_i to a_{i+1} for $1 \le i \le k$, where the indices are evaluated modulo k, so a_{k+1} equals a_1.

With slight abuse of notation, a cycle σ is also used to refer to the permutation that has σ as a cycle and all its other cycles are of length one. In this sense, the composition (product) of cycles is well-defined: we can write every permutation as a product of its (disjoint) cycles. This represents the cycle representation of a permutation. For example, as shown in Fig. 5(a), the cycle representation of $\pi = 23154$ is $(123)(45)$ and thus we may write $\pi = 23154 = (123)(45)$.

A cycle of length two is called a *transposition*. A transposition decomposition τ (or simply a decomposition) of a permutation π is a sequence $t_m \cdots t_1$ of transpositions t_i whose product is π. A decomposition of length m is called an m—decomposition. Note that the transpositions are applied from right to left.

A *sorting* s of a permutation π is a sequence of transpositions that transform π into $\imath$, where $\imath$ denotes the identity element of $\mathbb{S}_n$. In other words, $s\pi = \imath$. Note that a decomposition τ in reverse order equals a sorting s of the same permutation.

An *embedding* is a drawing of a graph such that no two edges cross. An embedding of $\mathcal{G}(\pi)$ can be obtained by placing vertices of disjoint cycles on disjoint circles as seen in Fig. 5(a). This embedding is also referred to as $\mathcal{G}(\pi)$. If the direction of the edges of $\mathcal{G}(\pi)$ are not explicitly indicated, we assume a clockwise direction and treat $\mathcal{G}(\pi)$ as a non-directional graph.

For a decomposition $\tau = t_m \cdots t_1$, let $\mathcal{T}(\tau)$ be a (multi)graph with vertex set $[n]$ and edges $(a_i b_i)$ for each transposition $t_i = (a_i b_i)$ of τ. We use the words transposition and edge interchangeably. The embedding of $\mathcal{T}(\tau)$ with vertex set $[n]$ into $\mathcal{G}(\pi)$ is also denoted by $\mathcal{T}(\tau)$. An example is shown in Fig. 5(b).

A permutation π is said to be odd (even) if the number of pairs $a, b \in [n]$ such that $a < b$ and $\pi(a) > \pi(b)$ is odd (even). If a permutation is odd (even), then the number of transpositions in any of its decompositions is also odd (even).

The following definitions regarding graphs $G = (V, E)$ will be used throughout the paper. An edge with endpoints u and v is denoted by $(uv) \in E$. A graph is said to be planar if it has an embedding. The subgraph of G induced by the vertices in the set $S \subset V$ is denoted by $G[S]$. The degree of a vertex v in G is denoted by $\deg_G(v)$ or, if there is no ambiguity, by $\deg(v)$. Deletion of an edge e from a graph G is denoted by $G - e$ and deletion of a vertex v and its adjacent edges from G is denoted by $G - v$. The same notions can be defined for multigraphs—graphs in which there may exist multiple edges between two vertices.

We say that an edge e in G is a cut edge for two vertices a and b, denoted by a, b-cut edge, if in $G - e$ there exists no path between a and b. The well known Menger's theorem [25] asserts that the minimum number of edges one needs to delete from G to disconnect a from b is also the maximum number of pairwise edge-disjoint paths between a and b. This theorem holds for multigraphs as well.

In the derivations to follow, we make frequent use of the spanning trees of the (multi)graphs $\mathcal{T}(\tau)$, $\mathcal{G}(\pi)$ and $\mathcal{G}(\pi) \cup \mathcal{T}(\tau)$. A spanning tree is a standard notion in graph theory: it is a tree that contains all vertices of the underlying (multi)graph.

We are concerned with the following problem: given a nonnegative cost function φ on the set of transpositions, the cost of a transposition decomposition is defined as the sum of costs of its transpositions. The task is to find an efficient algorithm for generating the Minimum Cost transposition Decomposition (MCD) of a permutation $\pi \in \mathbb{S}_n$. The cost of the MCD of a permutation π under cost function φ is denoted by $M_\varphi(\pi)$. If there is no ambiguity, the subscript is omitted.

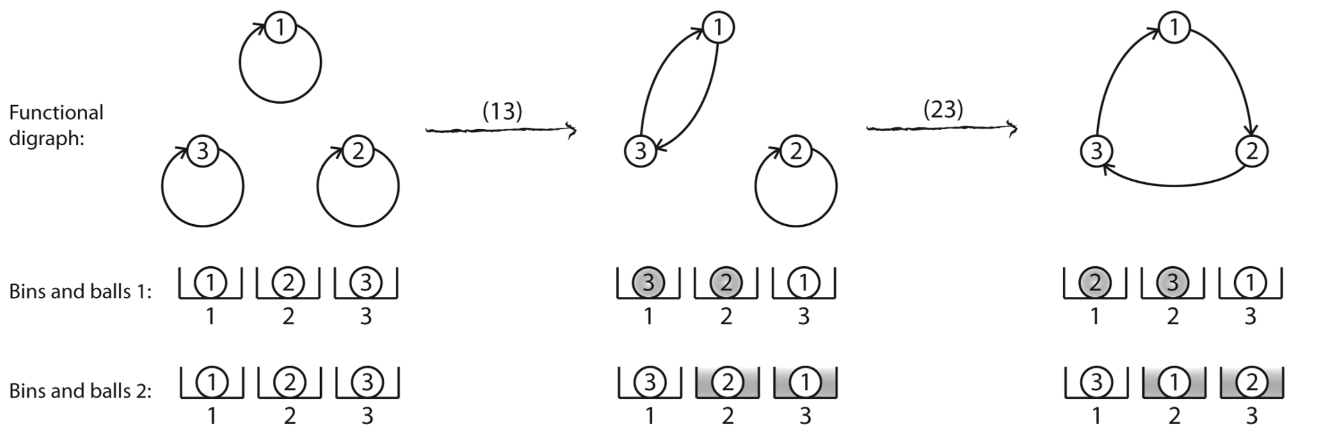

Fig. 6. The first row shows three functional digraphs: on the left, the functional digraph of the identity permutation, in the middle the digraph of (13), and on the right, the digraph of $(23)(13) = (123)$. The second and the third rows show the two binning models of the first row. The second row assumes i is mapped to j if ball i is in bin j. In the third row, the mapping is interpreted so that i is mapped to j if ball j occupies bin i.

For a non-negative cost function φ, let $\mathcal{K}(\varphi)$ be the undirected complete graph in which the cost of each edge (ab) equals $\varphi(a,b)$. The cost of a graph $G \subseteq \mathcal{K}(\varphi)$ is the sum of the costs of its edges

$$\text{cost}\,(G) = \sum_{(ab)\in G} \varphi(a,b).$$

The shortest path, i.e., the path with minimum cost, between i and j in $\mathcal{K}(\varphi)$ is denoted by $p^*(i,j)$.

The following definitions pertaining to cost functions are useful in our analysis. A cost function φ is a metric if for a, b, $c \in [n]$

$$\varphi(a,c) \le \varphi(a,b) + \varphi(b,c).$$

A cost function φ is a *metric-path* cost if it is defined in terms of a weighted path, denoted by Θ_s. The weights of edges (uv) in Θ_s are equal to $\varphi(u,v)$, and the cost of any transposition (ij) equals

$$\varphi(i,j) = \sum_{t=1}^{l} \varphi(c_t, c_{t+1}),$$

where $c_1 \cdots c_l c_{l+1}, c_1 = i, c_{l+1} = j$, represents the unique path between i and j in Θ_s. The path Θ_s is called the *defining path* of φ. A cost function φ is an *extended-metric-path* cost function if for a defining path Θ_s, $\varphi(i,j)$ is finite only for the edges (ij) of the defining path, and unbounded otherwise.

The definitions of permutations, cycles, and transpositions are stated above in terms of functions and graphs, or within an algebraic context. Two other ways to view permutations are in terms of binning schemes (bins and balls models).

Consider a permutation π of $[n]$, corresponding to n balls labeled from 1 to n, and n bins, labeled in the same way.

1) In the first representation, $\pi(i) = j$ if bin i is occupied by ball j. In other words, in this representation, the permutation describes the occupancies of bins. Applying a transposition (ab) to a permutation is equivalent to moving ball a to the bin occupied by ball b and vice versa. Accordingly, the cost $\varphi(a,b)$ can be seen to depend only on the balls and not on their location. This representation corresponds to the one-line notation described above.
2) Alternatively, the permutation can indicate the locations of balls. That is, $\pi(i) = j$ if ball i is in bin j. Here, applying a transposition (ab) is equivalent to swapping the contents of bins a and b. In this case, $\varphi(a,b)$ depends on the locations of the balls rather than on which balls are being moved.

An example of a permutation and its binning models is presented in Fig. 6.

The first representation allows us to apply the results of this paper to situations where the cost function depends on the elements being swapped and the second representation allows us to use the results in scenarios where the cost of swapping two elements depends on their location.

From an analytical point of view, although the first representation is more familiar because of its close relationship to the one-line notation, the second one, where $\pi(i)$ indicates the bin occupied by ball i, is more useful, and unless otherwise stated we refer to this representation simply as the balls and bins representation.

As an application of the second representation, consider a decomposition $\tau = t_m \cdots t_1$ of π and the graph $\mathcal{T}(\tau)$. Initially, imagine that the bins are located on the vertices $1, 2, \ldots, n$ of the graph, and each ball is in the bin with the same label. This configuration corresponds to the identity permutation. Consider the sequence of transpositions $t_1, t_2, \ldots, t_m$: first apply t_1, then t_2, and so on. Through this sequence of transpositions, the identity permutation is transformed into π, wherein ball i is in bin $\pi(i)$. In step j, in which the transposition $t_j = (x_j y_j)$ is applied, the balls placed in bins x_j and y_j, moving along the edge $(x_j y_j)$, swap their locations. Note that the balls "move" solely along the edges of $\mathcal{T}(\tau)$. Thus, there exists a path from i to $\pi(i)$ in $\mathcal{T}(\tau)$.

The predecessor of $a \in [n]$ in a permutation π is $\pi^{-1}(a)$ and the successor of a is $\pi(a)$. There is an edge from the predecessor of a to a and an edge from a to its successor in $\mathcal{G}(\pi)$. Applying a transposition (ab) to a permutation π is equivalent to exchanging the predecessors of a and b in $\mathcal{G}(\pi)$.

We define a generalization of the notion of a transposition, termed *h-transposition*, where the predecessor of a can be

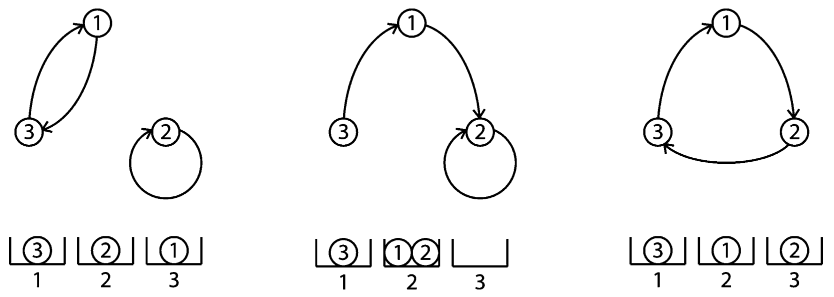

Fig. 7. The transposition (23) is equivalent to the product $(2,(2\to3))(1,(3\to2))$ of h-transpositions. The first row shows the functional digraph of this operation and the second row shows its bins and balls representation.

changed independently of the predecessor of b. For example, let a, b, c, $d\in[n]$ and let $\pi(c)=a$ and $\pi(d)=b$. Let $\pi'=(c,(a\to b))\,\pi$, where $(c,(a\to b))$ denotes what we call an h-transposition. This h-transposition takes c, the predecessor of a, to b, without modifying the predecessor of b. That is, we have a mapping in which $\pi'(c)=\pi'(d)=b$, and a has no predecessor. Note that π' is no longer a bijection, and several elements may be mapped to one element.

A transposition represents the product of a pair of h-transpositions, as in

$$(ab)\pi=\left(\pi^{-1}(a),(a\to b)\right)\left(\pi^{-1}(b),(b\to a)\right)\pi.$$

Fig. 7 illustrates this fact, using both the functional digraph and the balls and bins representation. In the latter case, $(1,(3\to2))$ indicates that ball 3 is moved from bin 2 to bin 1 and $(2,(2\to3))$ indicates that ball 2 is moved from bin 2 to bin 3.

An *h-decomposition* h of a permutation π is a sequence of h-transpositions such that $h\imath=\pi$. We assign half the cost $\varphi(a,b)$ of a transposition (ab) to the h-transposition $(c,(a\to b))$. Note that the cost of $(c,(a\to b))$ depends only on a and b. It is clear that the minimum cost h-decomposition has cost less than or equal to the minimum cost of a transposition decomposition.

For a permutation π and a transposition (ab), it can be easily verified that $(ab)\pi$ consist of one more (or one less) cycle than π if and only if a and b are in the same cycle (in different cycles). Since the identity permutation has n cycles, a Minimum Length transposition Decomposition (MLD) of π has length $n-\ell$, where ℓ denotes the number of cycles of π. The minimum cost of an MLD of π, with respect to cost function φ, is denoted by $L_\varphi(\pi)$. For example, $(132)\,(45)=(45)\,(23)\,(12)$ is decomposed into three $(=5-2)$ transpositions. In particular, if π is a single cycle, then the MLD of the cycle has length $n-1$. A cycle of length k has k^{k-2} MLDs [26]. An MCD is not necessarily an MLD, as illustrated by the following example [27].

Example 1: Consider the cycle $\sigma=(1\cdots5)$ with $\varphi(i,j)=3$, for $|i-j|=1$, and $\varphi(i,j)=1$ otherwise (recall that all the indices are taken modulo the length of the cycle, which in this case equals five). It is easy to verify that the decomposition $(13)(14)(24)(35)(13)(14)$ is an MCD of σ with cost six, i.e., $M(\sigma)=6$. However, as we shall see later, the cost of a minimum cost MLD is eight, i.e., $L(\sigma)=8$. One such MLD is $(14)(23)(13)(45)$.

Our approach to finding the minimum cost decomposition of a permutation consists of three stages:

1) First, we find the minimum cost decomposition for each individual transposition. In particular, we show that the minimum cost decomposition of a transposition can be obtained by recursively substituting transpositions with triples of transpositions. This step is superfluous for the case when the cost function is a metric.
2) In the second step, we consider cycles only and assume that each transposition cost is optimized. Cycles have the simplest structure among all permutations, and furthermore, each permutation is a collection of cycles. Hence, several approximation algorithms operate on individual cycles and combine their decompositions. As part of this line of results, we describe how to find the minimum cost MLD and show that its cost is not more than a constant factor higher than that of the corresponding MCD. We also present a particularly simple-to-implement class of decompositions whose costs lie between the cost of a minimum MLD and a constant multiple of the cost of an MCD.
3) We generalize the results obtained for single cycles to permutations with multiple cycles.

IV. Optimizing Individual Transposition

In this section, we find the MCD of transpositions, i.e., permutations of the form $\pi=(ab)$ for a, $b\in[n]$ and $a\neq b$. In the context of our problem, these are the simplest permutations. It is clear that the unique one-decomposition of $\pi=(ab)$ is (ab). Since π is odd, it has no two-decompositions. For $c\in[n]\backslash\{a,b\}$,

$$(ac)(bc)(ac),\tag{1}$$

is a three-decomposition of π. It is straightforward to see that any three-decomposition of π must be of the form (1), with a possible exchange of the roles of the elements a and b.

If $\varphi(a,b)>2\varphi(a,c)+\varphi(b,c)$, then $(ac)(bc)(ac)$ is a decomposition of π with smaller cost than $\varphi(a,b)$. Furthermore, one may substitute (ac) by, say, $(cd)(ad)(cd)$ to obtain a decomposition with yet smaller cost if $\varphi(a,c)>2\varphi(c,d)+\varphi(a,d)$. The same procedure may be repeated until it is no longer possible to reduce the cost further. The resulting decomposition is called a *triple-optimized decomposition*. It is straightforward to develop an algorithm that finds the triple-optimized decomposition for all transpositions. One such algorithm—Alg. 1—performs a simple search on the ordered set of transpositions in order to check if their product, of the form of (1), yields a decomposition of lower cost for some transposition. It then updates the costs of transpositions and performs a new search for decompositions of length three that may reduce some transposition cost.

Algorithm 1 OPTIMIZE-TRANSPOSITION-COSTS (Ω)

```
1: Input: $\Omega$ (the list of transpositions and their cost)
2: Sort $\Omega$
3: **for** $i \leftarrow 2, \dots, |\Omega|$ **do**
4: $(a_1 b_1) \leftarrow T\Omega(i)$
5: $\phi_1 \leftarrow \varphi(a_1, b_1)$
6: **for** $j = 1, \dots, i-1$ **do**
7: $(a_2 b_2) \leftarrow T\Omega(j)$
8: $\phi_2 \leftarrow \varphi(a_2, b_2)$
9: **if** $\{a_1 b_1\} \cap \{a_2 b_2\} \neq \{\}$ **then**
10: $a_{com} \leftarrow \{a_1, b_1\} \cap \{a_2, b_2\}$
11: $\{a_3, b_3\} \leftarrow \{a_1, a_2, b_1, b_2\} \backslash \{a_{com}\}$
12: **if** $\phi_1 + 2\phi_2 < \varphi(a_3, b_3)$ **then**
13: $\varphi(a_3, b_3) \leftarrow \phi_1 + 2\phi_2$
14: update $\varphi(a_3, b_3)$ in $\Omega$
15: Sort $\Omega$
```

The *triple-optimized costs* produced by the algorithm are denoted by φ^*. Note that $\varphi^*(a,b) \leq 2\varphi^*(a,x) + \varphi^*(b,x)$, for any $x \neq a, b$. Although an optimal decomposition of the form produced by Alg. 1 is not guaranteed to produce the overall minimum cost decomposition of any transposition, we show that this is indeed the case after the expositions associated with Alg. 1.

Observe that if the cost function is such that

$$\varphi(b,c) + 2\varphi(a,c) \geq \varphi(a,b), \quad a,b,c \in [n], \tag{2}$$

as in Example 1, Alg. 1 is redundant and can be omitted when computing the MCD. In particular, if the cost function is a metric, then Alg. 1 is not needed.

The input to Alg. 1 is an ordered list Ω of transpositions and their costs. Each row of Ω corresponds to one transposition and is of the form $[(ab)|\varphi(a,b)]$. The jth row of Ω is denoted by $\Omega(j)$ and the transposition corresponding to $\Omega(j)$ is denoted by $T\Omega(j)$. Sorting of Ω means reordering its rows so that transpositions are sorted in increasing order of their costs. The output of the algorithm is a list with the same format, but with triple-optimized costs for each transposition.

Lemma 2: Alg. 1 outputs the triple-optimized cost φ^* for all transpositions in $\mathbb{S}_n$.

Proof: Let Ω_i be the list Ω at the beginning of iteration i, obtained immediately before executing line 4 of Alg. 1. Also, let $\Omega_i(j,\dots,k)$ denote the rows $j, j+1, \dots, k$ of Ω_i and $T\Omega_i(j,\dots,k)$ denote the transpositions corresponding to these rows.

We prove, by induction, that $T\Omega_i(1,\dots,i)$ are triple-optimized and $\Omega_i(1,\dots,i)$ do not change in subsequent iterations of the algorithm, and that $\Omega_i(i+1,\dots,|\Omega|)$ are triple-optimized with respect to $\Omega_i(1,\dots,i-1)$, i.e., any triple-optimized decomposition of a transposition $(ab) \in T\Omega_i(i+1,\dots,|\Omega|)$ in terms of $T\Omega_i(1,\dots,i-1)$ has cost at least as large as the cost of (ab) in $\Omega_i(i+1,\dots,|\Omega|)$.

The claim is obviously true for $i=2$.

Assume that the claim holds for i. Note that, by the induction hypotheses, $\Omega_i(1,\dots,i) = \Omega_{i+1}(1,\dots,i)$ and that $T\Omega_i(i+1,\dots,|\Omega|) = T\Omega_{i+1}(i+1,\dots,|\Omega|)$. Let $(a_1b_1) = T\Omega_i(i)$ and consider a transposition $s \in T\Omega_{i+1}(i+1,\dots,|\Omega|)$. By the induction hypotheses, s is already triple-optimized with respect to $\Omega_{i+1}(1,\dots,i-1)$. Thus, the cost of s may be reduced using transpositions $T\Omega_{i+1}(1,\dots,i)$ only if one can write s as $(a_2b_2)(a_1b_1)(a_2b_2)$, where $(a_2b_2) \in T\Omega_{i+1}(1,\dots,i-1)$. But these are precisely the decompositions considered in the ith iteration and thus $\Omega_{i+1}(i+1,\dots,|\Omega|)$ is triple-optimized with respect to $\Omega_{i+1}(1,\dots,i)$.

Furthermore, $T\Omega_{i+1}(i+1)$ has the minimum cost among $T\Omega_{i+1}(i+1,\dots,|\Omega|)$ and thus its cost cannot be further reduced. Hence, the costs of transpositions $T\Omega_{i+1}(1,\dots,i+1)$ are triple-optimized and do not change in the subsequent iterations of the algorithm. ■

Example 3: The left-most list in (3) represents the input Ω to the algorithm, with transpositions in increasing order of their costs. This is equal to Ω_2. The two lists that follow represent updates of Ω produced by Alg. 1.

In the first iteration, the algorithm considers the transposition (13), for $i=2$, and the transposition (34), for $j=1$. Using these transpositions we may write $(34)\,(13)\,(34) = (14)$. The initial cost of (14) is 12 which exceeds $2\varphi(3,4) + \varphi(1,3) = 8$ and thus Ω_3 is obtained, i.e., the second list in (3).

Next, for $i=3$ and $j=1$, the algorithm considers (24) and (34). Since $(34)\,(24)\,(34) = (23)$, we update the cost of (23) from 23 to 11 as shown in the third list Ω_4 in (3). Additional iterations of the algorithm introduce no further changes in the costs.

$$\begin{bmatrix} (34) & 2 \\ (13) & 4 \\ (24) & 7 \\ (14) & 12 \\ (12) & 15 \\ (23) & 23 \end{bmatrix} \rightarrow \begin{bmatrix} (34) & 2 \\ (13) & 4 \\ (24) & 7 \\ \mathit{(14)} & \mathit{8} \\ (12) & 15 \\ (23) & 23 \end{bmatrix} \rightarrow \begin{bmatrix} (34) & 2 \\ (13) & 4 \\ (24) & 7 \\ (14) & 8 \\ \mathit{(23)} & \mathit{11} \\ (12) & 15 \end{bmatrix} \tag{3}$$

□

Computational Complexity: For each index i the number of operations performed in the algorithm is $O(|\Omega|)$. Thus, the total complexity of the algorithm is $O(|\Omega|^2)$. Since $|\Omega|$ is at most equal to the number of transpositions, we have $|\Omega| = \binom{n}{2}$. Hence, the complexity of Alg. 1 equals $O(n^4)$.

Since the transposition costs are arbitrary non-negative values, it is not clear that the minimum cost decomposition of a transposition is necessarily the triple-optimized cost obtained by Alg. 1. This algorithm only guarantees that one can identify the optimal sequence of consecutive replacements of transpositions by triples of transpositions. Hence, the minimum cost of a transposition (ab) may be smaller than $\varphi^*(a,b)$, i.e., there may be decompositions of length five, seven, or longer, which allow for an even smaller decomposition cost of a transposition.

Fortunately, this is not the case: we first prove this claim for decompositions of length five via exhaustive enumeration and then proceed to prove the general case via the use of Mengers's theorem for multigraphs [3]. We choose to provide the example of five-decompositions since it illustrates the difficulty of proving statements about non-minimum-length decompositions of permutations using exhaustive enumeration techniques. Graphical representations, on the other hand, allow for much more general and simpler proofs pertaining to non-minimum decompositions of transpositions.

We start by considering all possible transposition decompositions of length five, for which the transposition costs are first optimized via Alg. 1. In other words, we investigate if there exist decompositions of (ab) of length five that have cost smaller than $\varphi^*(a,b)$. Once again, observe that the costs of all transpositions used in such decompositions are first optimized via a sequence of triple-transposition decompositions. To reduce the number of cases to be investigated, we present the following lemma restricting the possible configurations in a multigraph corresponding to the decomposition of a transposition (ab).

Lemma 4: Let τ be a decomposition of a transposition (ab). The multigraph $\mathcal{M} = \mathcal{T}(\tau)$, where τ does not contain (ab), has the following properties:

1) The vertices a and b both have degree at least one.
2) The degree of at least one of the vertices a and b is at least two.
3) Every connected vertex of $\mathcal{M}$, other than a and b, appears in a closed walk. The closed walk may have repeated vertices but it cannot have any repeated edges.

Proof:

1) In order to swap a and b, both a and b must be moved.
2) If both vertices a and b have degree one, then a and b are moved exactly once. This is possible only if $(ab) \in \tau$.
3) Let $\tau = t_m \cdots t_2 t_1$. Let t_i be the transposition with the smallest index i that includes $x \in V(\mathcal{M})$. In the permutation $\tau_1 = t_i \cdots t_2 t_1$, x is not in its original location but rather occupies the position of another element, say, y. This means that there is a path from x to y in $\mathcal{T}(\tau_1)$. Similarly, there must exist a path from y to x in $\mathcal{T}(t_m \cdots t_{i+2} t_{i+1})$. Thus there is a closed walk with no repeated edges from x to itself in $\mathcal{M}$. ■

Let $x_1, x_2, \ldots, x_N$ be vertices included in the decomposition τ other than a and b. If $|E(\mathcal{M})|$ denotes the number of edges in the multigraph $\mathcal{M} = \mathcal{T}(\tau)$, then

$$2|E(\mathcal{M})| = \sum_{i=1}^{N} \deg(x_i) + \deg(a) + \deg(b).$$

From parts 1 and 2 of Lemma 4, it follows that $\deg(a) + \deg(b) \geq 3$ and, from part 3, one has that $\deg(x_i) \geq 2$. Hence, $2|E(\mathcal{M})| \geq 2N + 3$, and since N has to be an integer

$$N \leq \lfloor |E(\mathcal{M})| - \frac{3}{2} \rfloor = |E(\mathcal{M})| - 2. \quad (4)$$

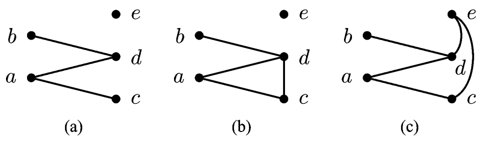

Fig. 8. Illustrations for scenario S1.

Suppose that $\tau = t_5 t_4 t_3 t_2 t_1$ is the minimum cost decomposition of (ab) with cost ϕ, and that the cost of the optimal decomposition produced by Alg. 1 exceeds ϕ. Then there is no vertex x such that

$$G_1 = \{(ax), (ax), (bx)\}$$

is a subset of edges in the multigraph $\mathcal{M}$ since, in that case,

$$\varphi^*(a,b) \leq 2\varphi^*(a,x) + \varphi^*(b,x) \leq \phi.$$

Also, there exists no pair of vertices x, y such that

$$G_2 = \{(ax), (bx), (ay), (by)\}$$

is a subset of edges in the multigraph $\mathcal{M}$. To prove this claim, suppose that $G_2 \subseteq E(\mathcal{M})$. Without loss of generality, assume that

$$\varphi^*(a,x) + \varphi^*(b,x) \leq \varphi^*(a,y) + \varphi^*(b,y).$$

Then,

$$\begin{aligned}\varphi^*(a,b) &\leq 2\varphi^*(a,x) + \varphi^*(b,x)\\ &\leq 2\varphi^*(a,x) + 2\varphi^*(b,x)\\ &\leq \phi.\end{aligned}$$

Hence, any decomposition of length five that contains G_2 must have cost at least $\varphi^*(a,b)$.

For any five-decomposition τ, we have $|E(\mathcal{M})| = 5$ and, thus, $N \leq 3$. We consider all five-decompositions of (ab) such that $\mathcal{M}$ is G_1—free and G_2—free, and that contain at most five vertices in $\mathcal{M}$. Assume that the three vertices of the graph, in addition to a and b, are denoted by c, d, and e. We now show that for each decomposition of length five, there exists a decomposition obtained via Alg. 1 with cost at most ϕ, denoted by either μ or μ'. The following scenarios are possible.

S1) Suppose that $\deg(a) = 2$ and $\deg(b) = 1$. Furthermore, suppose that there exist a vertex that is adjacent to both a and b in $\mathcal{M}$. Without loss of generality, assume that the neighbors of a and b are c and d, that is, $\mathcal{M}$ contains the graph of Fig. 8(a).

We consider two cases, depending on the existence of the edge (cd) in $\mathcal{M}$. First, assume that $(cd) \in \mathcal{M}$. In this case, $\mathcal{M}$ contains the graph of Fig. 8(b). If $\varphi^*(a,c) + \varphi^*(c,d) \leq \varphi^*(a,d)$, then the decomposition

$$\mu = (ac)(cd)(bd)(cd)(ac)$$

has cost at most ϕ. Note that μ can be obtained from Alg. 1, since

$$\mu = (ac)(bc)(ac) = (ab).$$

On the other hand, if $\varphi^*(a,c)+\varphi^*(c,d)>\varphi^*(a,d)$, then the decomposition $\mu'=(ad)(bd)(ad)$ has cost at most ϕ.

Next assume that $(cd)\notin\mathcal{M}$. Since both c and d each must lie on a cycle, the only possible decomposition of (ab) is the one with the graph shown in Fig. 8(c). Now, if $\varphi^*(ad)\le\varphi^*(d,e)+\varphi^*(e,c)+\varphi^*(a,c)$, then the decomposition

$$\mu=(ad)(bd)(ad)$$

has cost at most ϕ. On the other hand, if $\varphi^*(ad)>\varphi^*(d,e)+\varphi^*(e,c)+\varphi^*(a,c)$, then the decomposition

$$\mu'=(ac)(ec)(ed)(bd)(ed)(ce)(ac) \quad (5)$$

has cost at most ϕ. Note that μ' can be obtained from Alg. 1, since

$$\begin{aligned}\mu'&=(ac)(ec)(be)(ec)(ac)\\&=(ac)(cb)(ac)\\&=(ab).\end{aligned}$$

S2) Suppose that $\deg(a)=2$ and $\deg(b)=1$, but there is no vertex adjacent to both a and b. Without loss of generality, assume c and d are adjacent to a and e is adjacent to b, that is, $\mathcal{M}$ contains the graph of Fig. 9(a). Since c, d, and e each must lie on a cycle, one must include two more edges in the graph, as shown in Fig. 9(b). Since d and c have a symmetric role in the decomposition, we may without loss of generality, assume that $\varphi^*(a,c)+\varphi^*(c,e)\le\varphi^*(a,d)+\varphi^*(d,e)$. Let μ be equal to

$$\mu=(ac)(ce)(eb)(ce)(ac).$$

Similar to (5), it is easy to see that the cost of μ is at most ϕ and that it can be obtained from Alg. 1.

S3) Assume that $\deg(a)=\deg(b)=2$ ($\mathcal{M}$ contains the graph of Fig. 10(a)).

Since e and c must lie on a cycle, the fifth transposition in the decomposition must be (ec) (Fig. 10(b)). If $\varphi^*(a,d)+\varphi^*(b,d)\le\varphi^*(b,e)+\varphi^*(e,c)+\varphi^*(c,a)$, then the decomposition

$$\mu=(ad)(bd)(ad)$$

has cost at most ϕ. Otherwise, if $\varphi^*(a,d)+\varphi^*(b,d)>\varphi^*(b,e)+\varphi^*(e,c)+\varphi^*(c,a)$, the decomposition

$$\mu'=(ac)(ec)(be)(ce)(ac)$$

has cost at most ϕ. Note that both μ and μ' represent decompositions of the form optimized by Alg. 1.

S4) Suppose that $\deg(a)=3$, $\deg(b)=1$, and that all edges adjacent to a and b are simple (i.e., they are not repeated). Without loss of generality, assume that e is adjacent to both a and b ($\mathcal{M}$ contains the graph of Fig. 11(a)). One edge must complete cycles that include c, d, and e.

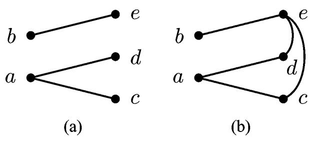

Fig. 9. Illustrations for scenario S2.

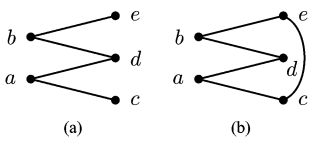

Fig. 10. Illustrations for scenario S3.

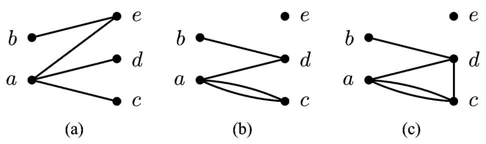

Fig. 11. Illustrations for scenarios S4 and S5.

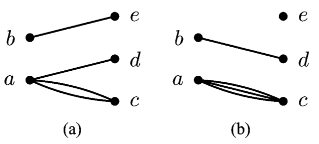

Fig. 12. Illustrations for scenario S6.

Since creating such cycles with one edge cannot be accomplished, this configuration is impossible.

S5) Suppose that $\deg(a)=3$, $\deg(b)=1$, one edge adjacent to a appears twice, and there is a vertex adjacent to both a and b. Without loss of generality, assume that this vertex is d ($\mathcal{M}$ contains the graph of Fig. 11(b)). Since d must be in a cycle, it must be adjacent to the "last edge", i.e., the fifth transposition. If the last edge is (ed), then one more edge is needed to create a cycle passing through e. Thus, the last edge cannot be (ed). The only other choice is (cd) and the corresponding graph $\mathcal{M}$ is shown in Fig. 11(c). Now, if $\varphi^*(a,d)\ge\varphi^*(c,d)$, then the decomposition

$$\mu=(ac)(cd)(bd)(cd)(ac)$$

has cost at most ϕ. Otherwise, if $\varphi^*(a,d)<\varphi^*(c,d)$, the decomposition

$$\mu'=(ad)(bd)(ad)$$

has cost at most ϕ.

S6) Suppose that $\deg(a)=3$, $\deg(b)=1$, and no vertex is adjacent to both a and b. Then $\mathcal{M}$ contains either the graph of Fig. 12(a) or that of Fig. 12(b). Since one edge cannot create all the necessary cycles, both configurations are impossible.

The Lemmas 5 and 6 are used in Theorem 7, which proves the optimality of Alg. 1.

Lemma 5: For a transposition (ab) and a decomposition τ of (ab), the graph $\mathcal{T}(\tau)$ has at most one a, b-cut edge.

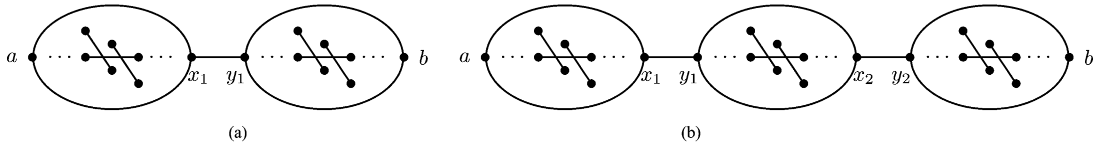

Fig. 13. Illustrations for the proof of Lemma 5: (a) A graph corresponding to a transposition decomposition of (ab) with one cut edge only. The cut edge is (x_1y_1) and it separates component B_1 containing a and component B_2 containing b. (b) A graph corresponding to a transposition decomposition of (ab) with two cut edges. The cut edges are (x_1y_1), which separates component B_1 containing a and component B_2 containing y_1, and (x_2y_2), which separates components B_2 and B_3 containing b.

Proof: Let $\mathcal{M} := \mathcal{T}(\tau)$. It can be easily shown that in $\mathcal{M}$, there exists a path between a and b. Consider the decomposition $t_m t_{m-1} \cdots t_i \cdots t_1$ of (ab) and suppose that $t_i = (x_1y_1)$ is an a, b-cut edge as shown in Fig. 13(a). The graph $\mathcal{M} - (x_1y_1)$ has two components; one contains a, denoted by B_1, and the other contains b, denoted by B_2. Since there exists a path from a to b, there also exists a path from a to x_1 which does not use the edge (x_1y_1). Thus, in $\mathcal{M} - (x_1y_1)$, a and x_1 are in B_1. Similarly, b and y_1 are in B_2. For $1 \le j \le m$, let $\pi_j = t_j \cdots t_1$. Since there is no transposition in π_{i-1} with endpoints in both B_1 and B_2, there is no element $z \in B_1$ such that $\pi_{i-1}(z) \in B_2$. Similarly, there is no element $z \in B_2$ such that $\pi_{i-1}(z) \in B_1$. This implies that $\pi_{i-1}(a) \in B_1$ and $\pi_{i-1}(b) \in B_2$. Since (x_1y_1) is the only edge connecting B_1 and B_2, we must have

$$\begin{aligned} \pi_{i-1}(a) &= x_1, & \pi_{i-1}(b) &= y_1, \\ \pi_i(a) &= y_1, & \pi_i(b) &= x_1. \end{aligned} \tag{6}$$

Also, for all $j \ge i$, we must have

$$\pi_j(a) \in B_2, \qquad \pi_j(b) \in B_1. \tag{7}$$

Alternatively, one can prove (6) and (7) by referring to the balls and bins model. Starting from the identity permutation, by applying the transpositions in $t_m t_{m-1} \cdots t_i \cdots t_1$, ball a is moved to bin b and vice versa while all other balls are in their original bins. Since (x_1y_1) is the only edge connecting the components B_1 and B_2, by applying $t_i = (x_1y_1)$, the balls a and b are moved from one component to the other. This implies (6). Furthermore, after applying (x_1y_1), ball a remains in B_2 and ball b remains in B_1, implying (7).

Now suppose there are at least two a, b-cut edge s in $\mathcal{M}$ as shown in Fig. 13(b). Let the decomposition of (ab) be $t_m \cdots t_l \cdots t_i \cdots t_1$, where $t_i = (x_1y_1)$ and $t_l = (x_2y_2)$ are a, b-cut edge s, for some $i < l$. Define B_1, B_2, and B_3 to be the components containing a, y_1, and y_2, respectively, in $\mathcal{M} - (x_1y_1) - (x_2y_2)$. By the same reasoning as above we must have

$$\pi_{l-1}(a) = x_2, \qquad \pi_{l-1}(b) = y_2.$$

However, this cannot be true since for all $j \ge i$, we have $\pi_j(b) \in B_1$ and thus $\pi_{l-1}(b) \ne y_2 \in B_3$. This contradiction shows that $\mathcal{M}$ cannot contain more than one a, b-cut edge. ■

Intuitively, one can interpret the cut edge result as follows. Every time a transposition is used its corresponding edge in $\mathcal{M}$ is deleted. If the position of a is changed through a sequence of transpositions including (x_1y_1), then edge (x_1y_1) is deleted and cannot be used in any sequence of transpositions that can potentially change the position of b to that of a. The only way to swap the location of a and b if there is one a, b-cut edge edge (x_1y_1) is to take a to the location of x_1 and b to the location of y_1. When there are two or more a, b-cut edge s, such swaps are impossible.

Lemma 6: For a, $b \in [n]$, a minimum cost subgraph $\mathcal{M}$ of $\mathcal{K}(\varphi)$ with at most one a, b-cut edge is a minimum cost subgraph among those consisting of two copies of a path between a and b with one edge deleted (removed).

Proof: First, suppose $\mathcal{M}$ has no a, b-cut edge. Then, by deleting edges from $\mathcal{M}$, we obtain a graph with cost smaller than or equal to the cost of $\mathcal{M}$ with one a, b-cut edge.

Thus, we may assume that $\mathcal{M}$ has exactly one a, b-cut edge. Suppose this a, b-cut edge is (xy) and a and x are in the same component of $\mathcal{M} - (xy)$. Then, there is no a, x-cut edge in $\mathcal{M}$, and thus, by Menger's theorem, there are two edge-disjoint paths between a and x. Delete the path with larger cost and instead add a copy of the path with smaller cost. Similarly, delete the path with larger cost between b and y and add a copy of the path with smaller cost. The graph obtained in this way has cost smaller than or equal to the cost of $\mathcal{M}$ and has only one a, b-cut edge. Furthermore, it consists of two copies of a path between a and b with one edge deleted. Namely, it has two copy of every edge in a path $a \cdots xy \cdots b$ except for (xy), of which it has only one copy. ■

Next, we state a general theorem pertaining to the optimality of Alg. 1.

Theorem 7: The minimum cost decompositions of all transpositions are generated by Alg. 1.

Proof: Consider the transposition (ab) and its MCD τ. Let $\mathcal{M} := \mathcal{T}(\tau)$. Let $\mathcal{M}'$ be a graph with minimum cost among those consisting of two copies of a path between a and b minus one edge. By Lemma 6, the cost of $\mathcal{M}'$ is less than or equal to the cost of $\mathcal{M}$. We show below that the cost obtained from Alg. 1 is, in turn, less than or equal to the cost of $\mathcal{M}'$, and by optimality of $\mathcal{M}$, this proves the theorem.

Suppose that $\mathcal{M}'$ consists of two copies of the path $ax_1x_2 \cdots x_r xyy_1y_2 \cdots y_s b$, for some integers r, s, minus one copy of (xy). Thus, we have that

$$\begin{aligned} C_1 := 2\varphi^*(a,x_1) + \cdots + 2\varphi^*(x_r,x) + \varphi^*(x,y) \ge \\ 2\varphi^*(a,x_1) + \cdots + 2\varphi^*(x_{r-1},x_r) + \varphi^*(x_r,y) \ge \\ 2\varphi^*(a,x_1) + \cdots + 2\varphi^*(x_{r-2},x_{r-1}) + \varphi^*(x_{r-1},y) \ge \\ \ge \cdots \ge \varphi^*(a,y) \end{aligned} \tag{8}$$

The cost of $\mathcal{M}'$ is $C_1+2\varphi^*(y,y_1)+2\varphi^*(y_1,y_2)+\cdots+2\varphi^*(y_sb)$ and for this we have

$$\begin{aligned}C_1 + 2\varphi^*(y,y_1) + 2\varphi^*(y_1,y_2) + \cdots + 2\varphi^*(y_s,b) \geq \\ \varphi^*(a,y) + 2\varphi^*(y,y_1) + \cdots + 2\varphi^*(y_s,b) \geq \\ \varphi^*(a,y_1) + 2\varphi^*(y_1,y_2) + \cdots + 2\varphi^*(y_s,b) \geq \\ \varphi^*(a,y_2) + 2\varphi^*(y_2,y_3) + \cdots + 2\varphi^*(y_s,b) \geq \\ \geq \cdots \geq \varphi^*(a,y_s) + 2\varphi^*(y_s,b) \geq \varphi^*(a,b) \quad (9)\end{aligned}$$

and thus the cost of the decomposition associated with $\mathcal{M}$, that is the MCD of (ab), cannot be smaller than the cost of the optimal decomposition produced by Alg. 1. Thus the cost obtained by Alg. 1 is the smallest possible. ■

The following lemma provides an alternative way of finding the MCD of individual transpositions.

Lemma 8: Among graphs consisting of two copies of a path between a and b minus one edge, the one with minimum cost is the graph $\mathcal{T}(\tau)$ for some MCD τ of (ab).

Proof: Let $\mathcal{M}'$ be a graph with minimum cost among those consisting of two copies of a path between a and b minus one edge and suppose that the path between a and b is $ax_1x_2\cdots x_rxyy_1y_2\cdots y_sb$. We show that there exists a decomposition τ of (ab) such that $\mathcal{M}'=\mathcal{T}(\tau)$.

Note that

$$\begin{aligned}(ab) = (ax_1\cdots x_rx)\,(by_s\cdots y_1y) \\ (xy)\,(yy_1\cdots y_sb)\,(xx_r\cdots x_1a)\,. \quad (10)\end{aligned}$$

Each of the cycles in (10) can be decomposed using the edges of the path between a and b as

$$\begin{aligned}(ax_1\cdots x_rx) &= (ax_1)\,(x_1x_2)\cdots(x_rx)\,,\\ (by_s\cdots y_1y) &= (by_s)\cdots(y_2y_1)\,(y_1y)\,,\\ (yy_1\cdots y_sb) &= (y_1y)\,(y_2y_1)\cdots(by_s)\,,\\ (xx_r\cdots x_1a) &= (x_rx)\cdots(x_1x_2)\,(ax_1)\,. \quad (11)\end{aligned}$$

Substituting the decompositions given in (11) in the right-hand side of (10) yields the desired transposition τ. ■

Lemma 8 indicates that to find an MCD of a transposition (ab), it is sufficient to find the path of the form described in the lemma. That is, we need to find

$$\arg\min_p \left(2\mathrm{cost}(p) - \max_{(xy)\in p} \varphi(x,y)\right)$$

where the minimum is taken over all paths between a and b in $\mathcal{K}(\varphi)$. Then, using (10) and (11), we can find an MCD of (ab). In the appendix, we describe a search algorithm based on this method, which is based on a modification of the well known Bellman-Ford procedure [28]. This approach should be compared to the approach of Alg. 1, which has the structure of a Viterbi-type search method for finding a minimum cost path in a transposition graph.

The following corollary, following from the preceding lemma, will be useful in Section V.

Corollary 9: For $a, b \in [n]$, the cost of an MCD of (ab) is less than or equal to twice the cost of any path p between a and b in $\mathcal{K}(\varphi)$. That is,

$$M_\varphi((ab)) = \varphi^*(a,b) \leq 2\mathrm{cost}(p).$$

Upon executing the algorithm, the cost of each transposition is set to the value found by Alg. 1. Only upon the completion of the last stage of the MCD approximation algorithm, to be presented in Sections V–VII, will each transposition be replaced by its minimum cost decomposition.

In summary, we showed that finding the MCD of transpositions is not a hard problem. Unfortunately, for permutations involving more than one transposition and for general non-negative cost functions, we cannot make the same statement and there is a possibility that the general problem is NP-hard. Section V, we take our analysis one step further and consider finding approximate solutions for individual cycles.

V. Optimizing Individual Cycles

We consider next the cost optimization problem over single cycles. First, we find the minimum cost MLD via a dynamic programming algorithm. The minimum cost MLD is obtained with respect to the optimized cost function φ^* of the previous section.

We also present a second algorithm to find decompositions whose cost, along with the cost of the minimum cost MLD, is not more than a constant factor higher than the cost of the MCD. Both algorithms are presented for completeness.

The results in this section apply to any cycle σ. However, for clarity of presentation, and without loss of generality, we consider the cycle $\sigma = (12\cdots k)$.

A. Minimum Cost, Minimum Length Transposition Decomposition

Recall that the vertices of $\mathcal{G}(\sigma)$ are placed on a circle. For an MLD τ of a permutation π with ℓ cycles, $\mathcal{T}(\tau)$ is a forest with ℓ components; each tree in the forest is the decomposition of one cycle of π. This can be easily seen by observing that each cycle corresponds to a tree. The following lemma provides a rigorous proof for this statement.

Lemma 10: The graph $\mathcal{T}(\tau)$ of an MLD τ of a cycle σ is a tree.

Proof: First, we show that $\mathcal{T}(\tau)$ is connected. In σ, $i \in [k]$ is the predecessor of $\sigma(i)$. Since $\sigma = \tau$, there exists a path, formed by a substring of transpositions of τ, between i and $\sigma(i)$ in $\mathcal{T}(\tau)$. Since there is a path from i to $\sigma(i)$, for $1 \leq i \leq k$, $\mathcal{T}(\tau)$ is connected.

To complete the proof, observe that $\mathcal{T}(\tau)$ has k vertices and $k-1$ edges since an MLD of a cycle of length k contains $k-1$ transpositions. Hence, $\mathcal{T}(\tau)$ is a tree. ■

For related ideas regarding permutation decompositions and graphical structures, the interested reader is referred to [29].

The following definitions will be used in the proof of Lemma 11, which states that $\mathcal{G}(\sigma) \cup \mathcal{T}(\tau)$ is planar, provided that τ is an MLD of σ.

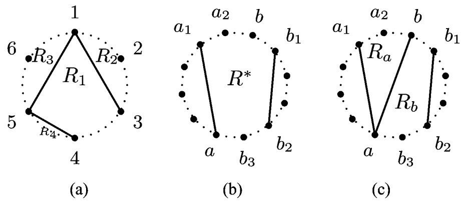

Fig. 14. (a) A tree T divides R into four subregions. (b,c) A region is divided into two regions by transposition (ab). See proof of Lemma 11. (a) T (b) $\tau_i\sigma$ (c) $\tau_{i+1}\sigma$.

Let R be the region enclosed by the edges of $\mathcal{G}(\sigma)$. Also, let T be a tree whose vertex set is a subset of $\{1,2,\ldots,k\}$, such that $\mathcal{G}(\sigma)\cup T$ is planar. Since T is a tree with edges contained in R, the edges of T partition R into smaller regions; each of these regions is the enclosure of a subset of edges of $\mathcal{G}(\sigma)\cup T$ and includes the vertices of these edges. The vertices $\{1,2,\ldots,k\}$ can be divided into *corner vertices*, lying at the intersection of at least two regions, and *inner vertices*, belonging only to one region. In Fig. 14(a), $\mathcal{G}(\sigma)$ with vertices $\{1,2,3,4,5,6\}$ is partitioned by T into four regions, R_1, R_2, R_3 and R_4. In R_2, vertices 1 and 3 are corner vertices, while vertex 2 is an inner vertex.

Lemma 11: For an MLD $\tau = t_1\cdots t_{k-1}$ of σ, $\mathcal{T}(\tau)\cup\mathcal{G}(\sigma)$ is planar. That is, for $t_i=(a_1a_2)$, where $a_1<a_2$, and $t_j=(b_1b_2)$, where $b_1<b_2$, if $a_1<b_1<a_2$, then $a_1<b_2<a_2$.

Proof: Note that $\tau^{-1}\sigma=\imath$. Let $\tau_i=t_{i-1}\cdots t_1$. Since τ is an MLD of σ, $\tau_i\sigma$ has i cycles. The proof proceeds by showing that for all $1\le i\le k$, the following two claims are true:

(I) $\mathcal{G}(\sigma)\cup\mathcal{T}(\tau_i)$ is planar.

(II) Each cycle of τ_i corresponds to a subregion R of $\mathcal{G}(\sigma)\cup\mathcal{T}(\tau_i)$. The cycle corresponding to R contains all of its inner vertices and some of its corner vertices but no other vertex.

Both claims (I) and (II) are obvious for $i=1$. We show that if (I) and (II) are true for τ_i, then they are also true for τ_{i+1}.

Let $t_i=(ab)$. Clearly, $\tau_{i+1}\sigma=t_i\tau_i\sigma$ has one more cycle than $\tau_i\sigma$, and by assumption, $\mathcal{G}(\sigma)\cup\mathcal{T}(\tau_i)$ is planar and partitioned into a set of subregions. Note that a and b are in the same cycle, and thus are inner or corner vertices of some subregion R^* of $\mathcal{G}(\sigma)\cup\mathcal{T}(\tau_i)$. The edge (ab) divides R^* into two subregions, R_a and R_b (without crossing any edge in $\mathcal{G}(\sigma)\cup\mathcal{T}(\tau_i)$). This proves (I).

Let the cycle corresponding to R^* be

$$\mu=(aa_1\cdots a_lbb_1\cdots b_{l'})$$

as seen in Fig. 14(b). Then,

$$(ab)\mu=(aa_1\cdots a_l)(bb_1\cdots b_{l'}).$$

Now the cycles $(aa_1\cdots a_l)$ and $(bb_1\cdots b_{l'})$ in τ_{i+1} correspond to subregions R_a and R_b, respectively, as seen in Fig. 14(c). This proves claim (II) since the cycle corresponding to each subregion contains all of its inner vertices and some of its corner vertices but no other vertex. ■

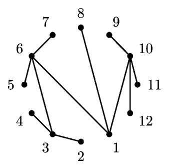

Fig. 15. Example illustrating the proof of Lemma 12: $r=10$, $s=8$.

The following lemma establishes a partial converse to the previous lemma.

Lemma 12: For a cycle σ and a spanning tree T over the vertices $\{1,2,\ldots,k\}$, if $\mathcal{G}(\sigma)\cup T$ is planar, then there exists at least one MLD τ of σ such that $T=\mathcal{T}(\tau)$.

Proof: We prove the lemma by recursively constructing an MLD corresponding to T. If $k=2$, then T has exactly one edge and the MLD is the transposition corresponding to that edge. For $k>2$, some vertex has degree larger than one. Without loss of generality, assume that $\deg(1)>1$. Let

$$r=\max\{u|\,(1u)\in T\}.$$

Since T is a tree, $T-(1r)$ has two components. These two components have vertex sets $\{1,\ldots,s\}$ and $\{s+1,\ldots,k\}$, for some s. It is easy to see that

$$(1\cdots k)=(s+1\cdots k1)(1\cdots s). \tag{12}$$

Let

$$\begin{aligned}T'&=T[\{1,\ldots,s\}],\\ T''&=T[\{s+1,\ldots,k,1\}].\end{aligned}$$

Note that T' and T'' have fewer than k vertices. Furthermore, $T'\cup\mathcal{G}((1\cdots s))$ and $T''\cup\mathcal{G}((s+1\cdots k1))$ are planar. Thus, by the induction hypothesis, $(1\cdots s)$ and $(s+1\cdots k1)$ have decompositions τ' and τ'' of length $s-1$ and $k-s$, respectively. By (12), $\tau''\tau'$ is an MLD for σ. ■

Example 13: In Fig. 15, we have $r=10$ and $s=8$. The cycle $(1\cdots 12)$ can be decomposed into two cycles

$$(1\cdots 12)=(9\ 10\ 11\ 12\ 1)(12\cdots 8).$$

Now, each of these cycles is decomposed in a similar manner into shorter cycles, for example,

$$\begin{aligned}(9\ 10\ 11\ 12\ 1)&=(9\ 10)(10\ 11\ 12\ 1),\\ (1\cdots 8)&=(8\ 1)(1\cdots 7).\end{aligned}$$

The same type of decomposition can be performed on cycles $(10\ 11\ 12\ 1)$ and $(1\cdots 7)$. □

Since any MLD of a cycle can be represented by a tree that is planar on the circle, the search for an MLD of minimum cost only needs to be performed over the set of planar trees. This search can be executed using a dynamic program, outlined in Alg. 2. Lemma 14 establishes that Alg. 2 produces a minimum cost MLD.

Algorithm 2 MIN-COST-MLD

```
1: Input: Optimized transposition cost function $\Phi^*$ where $\Phi^*_{i,j} = \varphi^*(i,j)$ (Output of Alg. 1)
2: $C(i,j) \leftarrow \infty$ for $i,j \in [k]$
3: $C(i,i) \leftarrow 0$ for $i \in [k]$
4: $C(i,i+1) \leftarrow \varphi^*(i,i+1)$ for $i \in [k]$
5: for $l = 2 \cdots k-1$ do
6:   for $i = 1 \cdots k-l$ do
7:     $j \leftarrow i+l$
8:     for $i \le s < r \le j$ do
9:       $A \leftarrow C(i,s) + C(s+1,r) + C(r,j) + \varphi^*(i,r)$
10:      if $A < C(i,j)$ then
11:        $C(i,j) \leftarrow A$
```

Lemma 14: The output cost of Alg. 2, $C(1,k)$, equals $L(\sigma)$.

Proof: The algorithm finds the minimum cost MLD of $(1\cdots k)$ by first finding the minimum cost of MLDs of shorter cycles of the form $(i\cdots j)$, where $1 \le i < j \le k$. We look at the computations performed in the algorithm from a top-down point of view.

Let $C_T(i,j)$ be the cost of the decomposition of the cycle $\sigma^{i,j} = (i\cdots j)$, using edges of $T[\{i,\ldots,j\}]$, where T is an arbitrary planar spanning tree over the vertices $\{1,\ldots,k\}$ arranged on a circle. For a fixed T, let r and s be defined as in the proof of Lemma 12. We may write

$$(i\cdots j) = (s+1\cdots r)(ir)(r\cdots j)(i\cdots s) \tag{13}$$

where $i \le s < r \le j$. Thus,

$$C_T(i,j) = C_T(s+1,r) + \varphi^*(i,r) + C_T(r,j) + C_T(i,s). \tag{14}$$

Define $C(i,j) = C_{T^*}(i,j)$, where

$$T^* = \arg\min_T C_T(i,j)$$

denotes a tree that minimizes the cost of the decomposition of $(i\cdots j)$. Then, we have

$$C(i,j) = C(s^*+1,r^*) + \varphi^*(i,r^*) + C(r^*,j) + C(i,s^*), \tag{15}$$

where s^* and r^* are the values that minimize the right-hand-side of (14) under the constraint $1 \le i \le s < r \le j$. Since the cost of each cycle can be computed from the cost of shorter cycles, $C(i,j)$ can be obtained recursively, with initialization

$$C(i,i+1) = \varphi^*(i,i+1). \tag{16}$$

The algorithm searches over s and r and computes $C(1,k)$ using (15) and (16).

Although these formulas are written in a recursive form, Alg. 2 is described as a dynamic program. The algorithm first computes $C(i,j)$ for small values of i and j, and then finds the cost of longer cycles. That is, for each $2 \le l \le k-1$ in increasing order, $C(i,i+l)$ is computed by choosing its optimal decomposition in terms of costs of smaller cycles. ■

Example 15: As an example, let us find the minimum cost decomposition of the cycle $\sigma = (1234)$ using the above algorithm. Let Φ be the matrix of transposition costs, with $\Phi_{ij} = \varphi(i,j)$:

$$\Phi = \begin{bmatrix} 0 & 5 & 10 & 3 \\ - & 0 & 2 & 3 \\ - & - & 0 & 9 \\ - & - & - & 0 \end{bmatrix},$$

$$\Phi^* = \begin{bmatrix} 0 & 5 & 9 & 3 \\ - & 0 & 2 & 3 \\ - & - & 0 & 7 \\ - & - & - & 0 \end{bmatrix} \tag{17}$$

After optimizing the transposition costs in Φ via Alg. 1, we obtain Φ^*, shown beneath Φ. From Alg. 2, we obtain

$$C(1,3) = C(2,3) + \varphi^*(1,2) = 7, (s,r) = (1,2),$$
$$C(2,4) = C(2,3) + \varphi^*(2,4) = 5, (s,r) = (3,4).$$

Consider the cycle (1234), where $i = 1$ and $j = 4$. The algorithm compares $\binom{4}{2} = 6$ ways to represent the cost of this cycle using the cost of shorter cycles. The minimum cost is obtained by choosing $s = 2$ and $r = 4$, so that

$$C(1,4) = C(2,4) + \varphi^*(1,4) = 8.$$

Writing C as a matrix, where $C(i,j) = C_{ij}$, we have

$$C = \begin{bmatrix} 0 & 5 & 7 & 8 \\ - & 0 & 2 & 5 \\ - & - & 0 & 7 \\ - & - & - & 0 \end{bmatrix}$$

Note that we can modify the above algorithm to also find the underlying MLD by using (13) to write the decomposition of every cycle with respect to r and s that minimize the cost of the cycle. For example, from (13), by substituting the appropriate values of r and s, we obtain

$$\begin{aligned}(1234) &= (234)(14) \\ &= (34)(24)(14).\end{aligned}$$

□

The initialization steps are performed in $O(k)$ time. The algorithm performs a constant number of steps for each i, j, r, and s such that $1 \le i \le s < r \le j \le k$. Hence, the computational cost of the algorithm is $O(k^4)$.

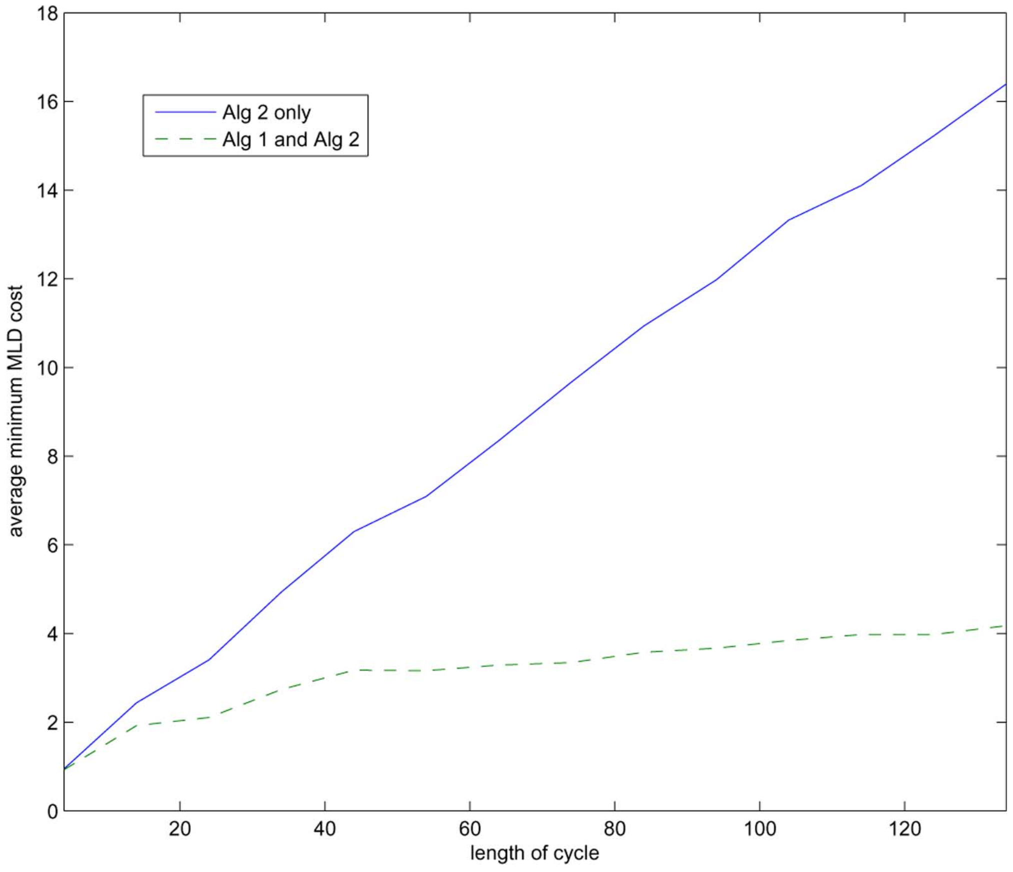

Fig. 16. The average minimum MLD cost vs the length of the cycle. Transposition costs are chosen independently and uniformly in $[0,1]$.

Note that Alg. 2 operates on the optimized cost function φ^*, obtained as the output of Alg. 1. Fig. 16 illustrates the importance of first reducing individual transposition costs using Alg. 1 before applying the dynamic program. Since the dynamic program can only use $k-1$ transpositions of minimum cost, it cannot optimize the individual costs of transpositions and strongly relies on the reduction of Alg. 1 for producing low cost solutions. In Fig. 16, the transposition costs were chosen independently according to a uniform distribution over $[0,1]$.

B. Constant-Factor Approximation for Cost of MCD

For the cycle $\sigma=(12\cdots k)$ and $1\le j\le k$, consider the decomposition

$$\begin{aligned}(j+1\,j+2)\,(j+2\,j+3)\cdots\\ (k-1\,k)\,(k1)\,(12)\,(23)\cdots(j-1\,j)\,.\end{aligned} \quad (18)$$

The cost of this decomposition equals $\sum_{i\in\sigma}\varphi^*(i,\sigma(i))-\varphi^*(j,\sigma(j))$.

To minimize the cost of the decomposition, we choose j such that the transpositions $(j\;j+1)$ has maximum cost. That is, we let $j=j^*$ in (18) where $j^*=\arg\max_{j\in\sigma}\varphi^*(j,\sigma(j))$. This decomposition is termed the Simple Transposition Decomposition (STD) of σ and its cost is denoted by $S(\sigma)$.

Theorem 16: For a cycle σ, $M(\sigma)\le L(\sigma)\le S(\sigma)\le 4M(\sigma)$.

Proof: Clearly, $M(\sigma)\le L(\sigma)$. It is easy to see that an STD is itself an MLD and, thus, $L(\sigma)\le S(\sigma)$. For $S(\sigma)$, we have

$$\begin{aligned}S(\sigma)&=\sum_{i\in\sigma}\varphi^*(i,\sigma(i))-\varphi^*(j^*,\sigma(j^*))\\ &\le\sum_{i\in\sigma}\varphi^*(i,\sigma(i))\le 2\sum_{i\in\sigma}\mathrm{cost}\,(p^*(i,\sigma(i)))\end{aligned} \quad (19)$$

where the last inequality follows from Lemma 8. To complete the proof, we need to show that $M(\sigma)\ge\frac{1}{2}\sum_i\mathrm{cost}\,(p^*(i,\sigma(i)))$. Since this result is of independent importance, we state it in Lemma 17. ■

Recall from Section III that

$$\left(\sigma^{-1}(a),(a\to b)\right)\left(\sigma^{-1}(b),(b\to a)\right)\sigma=(ab)\sigma$$

and that the cost of each h-transposition is half the cost of the corresponding transposition.

Lemma 17: It holds that $M(\sigma)\ge\frac{1}{2}\sum_i\mathrm{cost}\,(p^*(i,\sigma(i)))$.

Proof: Any decomposition can be written as an h-decomposition with the same cost by breaking each transposition into two h-transpositions. Thus, the minimum cost of a decomposition, $M(\sigma)$, is at least as large as the minimum cost of an h-decomposition. The minimum cost h-decomposition uses the shortest path $p^*(i,\sigma(i))$ between i and $\sigma(i)$. In this case, i becomes the predecessor of $\sigma(i)$ through the following sequence of h-transpositions:

$$(i,(v_m\to\sigma(i)))\cdots(i,(v_1\to v_2))\,(i,(i\to v_1)),$$

where $p^*(i,\sigma(i)) = iv_1v_2\cdots v_m\sigma(i)$ is the shortest path between i and $\sigma(i)$. This h-decomposition has cost

$$\frac{1}{2}\sum_i \mathrm{cost}\left(p^*(i,\sigma(i))\right),$$

and this completes the proof. ■

Observe that Theorem 16 asserts that a minimum cost MLD never exceeds the cost of the corresponding MCD by more than a factor of four. Hence, a minimum cost MLD represents a good approximation for an MCD, independent of the choice of the cost function. On the other hand, STDs are attractive alternatives to MLDs and dynamic programs, due to the fact that they are particularly simple to find algorithmically.

Example 18: Consider the cycle $\sigma = (12345)$ and the cost function φ, with $\varphi(2,4)=\varphi(2,5)=\varphi(3,5)=1$, and $\varphi(i,j)=100$ for all remaining transposition. From Lemma 17,

$$M(\sigma)\geq\frac{1}{2}(100+2+3+2+100)=103.5.$$

For example, the second term in the sum corresponds to a path going from 2 to 5 and then from 5 to 3. The cost of this path is two.

Since $M(\sigma)$ has to be an integer, it follows that $M(\sigma)\geq 104$. The optimized cost function φ^*, obtained from Alg. 1, equals

$$\varphi^*(i,j)=\begin{cases}1, & (ij)\in\{(25),(35),(24)\}\\ 3, & (ij)\in\{(23),(45)\}\\ 5, & (ij)=(34)\\ 100, & \text{otherwise}\end{cases}$$

A minimum cost MLD can be computed using the dynamic program of Alg. 2. One minimum cost MLD equals $\tau_L = (45)(35)(12)(25)$, and has cost $L(\sigma)=105$. By substituting each of the transpositions in τ_L with their minimum cost transposition decomposition, we obtain (24)(25)(24)(35)(12)(25).

It is easy to see that

$$\tau_s=(12)(23)(34)(45)$$

is an STD of σ with cost $S(\sigma)=100+3+5+3=111$.

Hence, the inequality $M(\sigma)\leq L(\sigma)\leq S(\sigma)\leq 4M(\sigma)$ holds. Furthermore, note that σ is an even cycle, and hence must have an even number of transpositions in any of its decompositions. This shows that $M(\sigma)=L(\sigma)=105$. □

C. Metric-Path and Extended-Metric-Path Cost Functions

We show next that for two non-trivial families of cost functions, one can improve upon the bounds of Theorem 16. For metric-path cost functions, a minimum cost MLD is actually an MCD, i.e., $L(\sigma)=M(\sigma)$. For extended-metric-path costs, it holds that $L(\sigma)\leq 2M(\sigma)$.

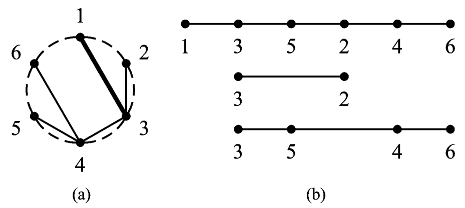

Fig. 17. Example illustrating the proof of Lemma 19: (a) The cycle $\sigma = (12345)$. Edges of $\mathcal{G}(\sigma)$ are shown with dashed arcs. The edge (13), shown with a thick solid line, belongs to $T(\sigma)$. The tree $T((3456))$ consists of solid edges on the left-hand side of (13) and $T((23))$ consists of solid edges on the right-hand side of (13). As stated in the proof, we have $T(\sigma)=(13)\cup T((23))\cup T((3456))$. (b) The defining path Θ_s of φ. Vertex 1 is a leaf and 3 is its parent. (a) $\mathcal{G}(\sigma)\cup T(\sigma)$ (b) Θ_s.

Note that metric-path costs are not the only cost functions which admit MCDs of the form of MLDs—another example includes star transposition costs. For such costs, one has $\varphi(i,j)=\infty$ for all i, j except for one index i. The remaining costs are arbitrary, but non-negative. The proof for this special case is straightforward and hence omitted.

Lemma 19: For a cycle σ and a metric-path cost function φ, $L(\sigma)\leq\frac{1}{2}\sum_i\varphi(i,\sigma(i))=\frac{1}{2}\sum_i\mathrm{cost}\left(p^*(i,\sigma(i))\right)$.

Proof: The equality in the lemma follows from the definition of metric-path cost functions.

We recursively construct a spanning tree $T(\sigma)$ with cost $B(\sigma)=\frac{1}{2}\sum_i\varphi(i,\sigma(i))$, such that $\mathcal{G}(\sigma)\cup T(\sigma)$ is planar. Since $T(\sigma)$ corresponds to an MLD, $L(\sigma)\leq B(\sigma)$. The validity of the recursive construction can be proved by induction.

For $k=2$, i.e., $\sigma=(12)$, $T(\sigma)$ is the edge (12) with cost $\varphi(1,2)=B(\sigma)$. Assume next that the cost of $T(\sigma)$ for any cycle σ of length $\leq k-1$ equals $B(\sigma)$.

For a cycle of length k, without loss of generality, assume that the vertex labeled 1 is a leaf in Θ_s (the defining path of φ) and that t is its parent. We construct $T(\sigma)$ from smaller trees by letting

$$T(\sigma)=(1t)\cup T((2\cdots t))\cup T((t\cdots k)).$$

See Fig. 17 for an illustration. The cost of $T(\sigma)$ is equal to $B((2\cdots t))+B((t\cdots k))+\varphi(1,t)$. Note that we can write

$$\begin{aligned}B((2\cdots t))&=\frac{1}{2}\sum_{i=2}^{t-1}\varphi(i,\sigma(i))+\frac{1}{2}\varphi(2,t)\\&=\frac{1}{2}\sum_{i=1}^{t-1}\varphi(i,\sigma(i))+\frac{1}{2}\varphi(2,t)-\frac{1}{2}\varphi(1,2),\\B((t\cdots k))&=\frac{1}{2}\sum_{i=t}^{k-1}\varphi(i,\sigma(i))+\frac{1}{2}\varphi(t,k)\\&=\frac{1}{2}\sum_{i=t}^{k}\varphi(i,\sigma(i))+\frac{1}{2}\varphi(t,k)-\frac{1}{2}\varphi(1,k).\end{aligned}$$

Since $\varphi(1,2)=\varphi(1,t)+\varphi(t,2)$ and $\varphi(1,k)=\varphi(1,t)+\varphi(t,k)$, it follows that

$$B((2\cdots t))+B((t\cdots k))+\varphi(1,t)=B((1\cdots k)).$$

This completes the proof of the lemma. ■

Theorem 20: For a cycle σ and a metric-path cost function, one has

$$L(\sigma) = M(\sigma) = \frac{1}{2}\sum_i \varphi\left(i, \sigma(i)\right).$$

Proof: Since $L(\sigma) \geq M(\sigma)$, it suffices to show that $L(\sigma) \leq \frac{1}{2}\sum_i \varphi(i,\sigma(i))$ and $M(\sigma) \geq \frac{1}{2}\sum_i \varphi(i,\sigma(i))$. Lemma 19 establishes that $L(\sigma) \leq \frac{1}{2}\sum_i \varphi(i,\sigma(i))$. From Lemma 17, it follows that

$$M(\sigma) \geq \frac{1}{2}\sum_i \text{cost}\left(p^*\left(i,\sigma(i)\right)\right).$$

Since φ is a metric-path cost function, we have $\varphi(i,\sigma(i)) = \text{cost}(p^*(i,\sigma(i)))$. This proves the claimed result. ■

Theorem 21: For extended-metric-path cost functions φ_e, $L_{\varphi_e}(\sigma) \leq 2M_{\varphi_e}(\sigma)$.

Proof: We prove the theorem by establishing that

$$L_{\varphi_e}(\sigma) \overset{(a)}{\leq} \sum_i \text{cost}\left(p^*\left(i,\sigma(i)\right)\right) \overset{(b)}{\leq} 2M_{\varphi_e}(\sigma)$$

where $p^*(i,\sigma(i))$ is the shortest path between i and $\sigma(i)$ in $\mathcal{K}(\varphi_e)$ and is calculated with respect to the cost function φ_e.

Let Θ_s be the defining path of an extended-metric-path cost φ_e. Consider the *metric-path cost* function, φ_m, with defining path Θ_s, and with costs of all edges $(ij) \in \Theta_s$ doubled. If the edge $(ij) \notin \Theta_s$, and if $c_1c_2\cdots c_{l+1}$ is the unique path from $c_1 = i$ to $c_{l+1} = j$ in Θ_s, then

$$\varphi_m(i,j) = \sum_{t=1}^{l} \varphi_m\left(c_t, c_{t+1}\right) = 2\sum_{t=1}^{l} \varphi_e\left(c_t, c_{t+1}\right).$$

By Corollary 9, $\varphi_e^*(i,j) \leq \varphi_m(i,j)$, for all i, j. Hence, $L_{\varphi_e}(\sigma) \leq L_{\varphi_m}(\sigma)$. Now, by Lemma 19,

$$\begin{aligned} L_{\varphi_e}(\sigma) &\leq L_{\varphi_m}(\sigma) \\ &= \frac{1}{2}\sum_i \varphi_m\left(i,\sigma(i)\right) \\ &= \sum_i \text{cost}\left(p^*\left(i,\sigma(i)\right)\right), \end{aligned} \tag{20}$$

which proves (a).

Note that Lemma 17 holds for all non-negative cost functions, including extended-metric-path cost functions. Thus,

$$M_{\varphi_e}(\sigma) \geq \frac{1}{2}\sum_i \text{cost}\left(p^*\left(i,\sigma(i)\right)\right),$$

which proves (b). ■

Example 22: Consider the cycle $\sigma = (12345)$ and the extended-metric-path cost function with $\Theta_s = 24135$ where the cost of each edge of Θ_s is 1.

By inspection, one can see that an MCD of σ is $(13)(14)(24)(35)(13)(14)$, with cost $M(\sigma) = 6$. A minimum cost MLD of σ is $(15)(34)(24)(14)$, with cost $L(\sigma) = 8$. The decomposition $(34)(45)(51)(12)$, is an STD with cost $S(\sigma) = 14$. Thus, we observe that the inequalities $L(\sigma) \leq 2M(\sigma)$ and $S(\sigma) \leq 4M(\sigma)$ are satisfied. □

In summary, we showed that the minimum cost MLD of a cycle can be obtained in polynomial time. Furthermore, we showed that the minimum cost MLD is a 4-approximation for the MCD. For some special cases for the choice of the cost function, the minimum cost MLD and the MCD coincide.

VI. Optimizing Permutations With Multiple Cycles

Most of the results in the previous section generalize to permutations with multiple cycles without much difficulty. We present next the generalization of those results.

Let π be a permutation in $\mathbb{S}_n$, with cycle decomposition $\sigma_1\sigma_2\cdots\sigma_\ell$. A decomposition of π with minimum number of transpositions is the product of MLDs of individual cycles σ_i. Thus, the minimum cost of an MLD of π equals

$$L(\pi) = \sum_{t=1}^{\ell} L\left(\sigma_t\right).$$

The STD of π is the product of the STDs of individual cycles σ_i.

The following theorem generalizes the results presented for individual cycles to permutations with multiple cycles.

Theorem 23: Consider a permutation π with cycle decomposition $\sigma_1\sigma_2\cdots\sigma_\ell$, and cost function φ. The following claims hold.

1) $S(\pi) \leq 2\sum_i \text{cost}(p^*(i,\pi(i)))$.
2) $M(\pi) \geq \frac{1}{2}\sum_i \text{cost}(p^*(i,\pi(i)))$.
3) $L(\pi) \leq S(\pi) \leq 4M(\pi)$.
4) If φ is a metric-path cost function, then
$$M(\pi) = L(\pi).$$
5) If φ is an extended-metric-path cost function, then
$$L(\pi) \leq 2M(\pi).$$

Proof:

1) For $1 \leq t \leq \ell$, it holds that
$$S\left(\sigma_t\right) \leq 2\sum_{i\in\sigma_t} \text{cost}\left(p^*\left(i,\sigma_t(i)\right)\right),$$
which can be seen by referring to (19) in the proof of Theorem 16. Thus,
$$\begin{aligned} S(\pi) &= \sum_{t=1}^{\ell} S\left(\sigma_t\right) \\ &\leq \sum_{t=1}^{\ell} 2\sum_{i\in\sigma_t} \text{cost}\left(p^*\left(i,\pi(i)\right)\right) \\ &= 2\sum_{i=1}^{n} \text{cost}\left(p^*\left(i,\pi(i)\right)\right). \end{aligned}$$
2) The same argument as in Lemma 17 applies without modifications.
3) For $1 \leq t \leq \ell$, from the proof of Theorem 16, we have $L(\sigma_t) \leq S(\sigma_t)$. Consequently,
$$L(\pi) = \sum_{t=1}^{\ell} L\left(\sigma_t\right) \leq \sum_{t=1}^{\ell} S\left(\sigma_t\right) = S(\pi).$$

Furthermore, from parts 1 and 2 of this theorem, it follows that $S(\pi) \leq 4M(\pi)$. Therefore $L(\pi) \leq S(\pi) \leq 4M(\pi)$.

4) From Lemma 19, for $1 \leq t \leq \ell$, it holds that

$$L(\sigma_t) \leq \frac{1}{2}\sum_{i\in\sigma_t} \text{cost}\left(p^*\left(i,\pi(i)\right)\right).$$

By summing over all cycles, we obtain

$$L(\pi) \leq \frac{1}{2}\sum_{i=1}^{n} \text{cost}\left(p^*\left(i,\pi(i)\right)\right).$$

The claimed result follows from part 2 and the fact that $M(\pi) \leq L(\pi)$.

5) From the proof of Theorem 21, we have $L(\sigma_t) \leq \sum_{i\in\sigma_t} \text{cost}\left(p^*\left(i,\pi(i)\right)\right)$. By summing over all cycles, we obtain

$$L(\pi) \leq \sum_{i=1}^{n} \text{cost}\left(p^*\left(i,\pi(i)\right)\right) \leq 2M(\pi),$$

where the last inequality follows from part 2 of this theorem.

■

A. Merging Cycles

In Section V, we demonstrated that the minimum cost of an MLD for a cycle represents a constant approximation for an MCD. The MLD of a permutation represents the product of the MLDs of individual cycles of the permutation. Clearly, optimization of individual cycle costs may not lead to the minimum cost decomposition of a permutation. For example, it may happen that the cost of transpositions within a cycle are much higher than the costs of transpositions between elements in different cycles. It is therefore useful to analyze how merging of cycles may affect the overall cost of a decomposition.

We propose a simple merging method that consists of two steps:

1) For a permutation π with k cycles, find a sequence of transpositions

$$\tau' = t_{k-1}\cdots t_1$$

so that $\sigma' = \tau'\pi$ is a single cycle. Ideally, this sequence should have minimum cost, although this is not required in the proofs to follow.

2) Find the minimum cost MLD τ of σ'.

The resulting decomposition of π is $\tau'^{-1}\tau$.

Each t_i is a transposition joining two cycles. The cost of τ' equals $\sum_{i=1}^{k-1}\varphi(a_i,b_i)$, where $t_i = (a_i b_i)$. The cost of the decomposition $\tau'^{-1}\tau'$ equals

$$C = \sum_{i=1}^{k-1}\varphi(a_i,b_i) + L(\sigma'). \tag{21}$$

Since $\sigma' = t_{k-1}\cdots t_1\pi$, we also have

$$L(\sigma') \leq 4M(\sigma') \leq 4\left(\sum_{i=1}^{k-1}\varphi(a_i,b_i) + M(\pi)\right). \tag{22}$$

Hence, from (21) and (22), C is upper bounded by

$$\begin{aligned} C &\leq \sum_{i=1}^{k-1}\varphi(a_i,b_i) + 4\left(\sum_{i=1}^{k-1}\varphi(a_i,b_i) + M(\pi)\right) \\ &\leq 5k\varphi_{max} + 4M(\pi), \end{aligned} \tag{23}$$

where φ_{max} is the highest cost in φ. The approximation ratio, defined as $C/M(\pi)$, is upper bounded by

$$\alpha \leq 4 + \frac{5k}{n-k}\frac{\varphi_{max}}{\varphi_{min}} = 4 + \frac{5k/n}{1-k/n}\frac{\varphi_{max}}{\varphi_{min}}, \tag{24}$$

which follows from the fact $M(\pi) \geq (n-k)\varphi_{min}$, where φ_{min} is the smallest cost in φ, assumed to be nonzero.

Although α is bounded by a value strictly larger than four, according to the expression above, this does not necessarily imply that merging cycles is sub-optimal compared to running the minimum cost MLD algorithm on individual cycles. Furthermore, if the MCDs of single cycles can be computed correctly, one can show that

$$\begin{aligned} C &= \sum_{i=1}^{k-1}\varphi(a_i,b_i) + M(\sigma'). \\ &\leq 2\sum_{i=1}^{k-1}\varphi(a_i,b_i) + M(\pi) \\ &\leq 2k\varphi_{max} + M(\pi) \end{aligned}$$

The approximation ratio in this case is upper bounded by

$$\alpha \leq 1 + \frac{2k}{n-k}\frac{\varphi_{max}}{\varphi_{min}} = 1 + \frac{2k/n}{1-k/n}\frac{\varphi_{max}}{\varphi_{min}}.$$

Lemma 24: Let π be a randomly chosen permutation from $\mathbb{S}_n$. Given that the MCDs of single cycles can be computed correctly, and provided that $\varphi_{max}/\varphi_{min} = o(n/\log n)$, α converges to one in probability as $n \to \infty$.

Proof: Let X_n be the random variable denoting the number of cycles in a random permutation $\pi_n \in \mathbb{S}_n$. It is well known that $EX_n = \sum_{j=1}^{n}\frac{1}{j} = H(n)$ and that $EX_n(X_n-1) = (EX_n)^2 - \sum_{j=1}^{n}\frac{1}{j^2}$ [30]. Here, $H(n)$ denotes the nth Harmonic number. Thus,

$$EX_n^2 = O\left((\ln n)^2\right), \tag{25}$$

which shows that $(X_n/n)(\varphi_{max}/\varphi_{min}) \to 0$ in quadratic mean as $n \to \infty$. Hence $(X_n/n)(\varphi_{max}/\varphi_{min}) \to 0$ in probability. By Slutsky's theorem [31], $\alpha \to 1$ in probability as $n \to \infty$. ■

In the following example, all operations are performed modulo 10, with zero replaced by 10.

Example 25: Consider the permutation $\pi = \sigma_1\sigma_2$, where $\sigma_1 = (1\,7\,3\,9\,5)$ and $\sigma_2 = (2\,8\,4\,10\,6)$, and the cost function φ,

$$\varphi(i,j) = \begin{cases} 1, & d(i,j) = 1 \\ \infty, & \text{otherwise} \end{cases}$$

where $d(i,j) = \min\{|i-j|, 10-|i-j|\}$.

Note that

$$\mathrm{cost}\,(p^*(i,j)) = d(i,j),$$

where p^* is the shortest path from i to j. We make the following observations regarding the decompositions of π.

1) MCD: Since we currently do not know of an efficient enough algorithm for finding the MCD of a permutation, we were not able to find the MCD of π. Nevertheless, using Theorem 23, one can obtain the following bound:

$$M(\pi) \geq \lceil \frac{1}{2}\sum_{i=1}^{10} \mathrm{cost}\,(p^*\,(i,\pi(i)))\rceil = \frac{2\cdot 5\cdot 4}{2} = 20.$$

2) MLD: As before, let the output of Alg. 1 be denoted by φ^*. We have $\varphi^*(i,j) = 2d(i,j) - 1$. The minimum cost MLDs for the cycles are

$$(1\,7\,3\,9\,5) = (1\,9)\,(3\,7)\,(1\,3)\,(9\,5)\,,$$
$$(2\,8\,4\,10\,6) = (2\,10)\,(4\,8)\,(2\,4)\,(6\,10)\,,$$

each of cost 20. A minimum cost MLD of π is the concatenation of the minimum cost MLDs of σ_1 and σ_2:

$$\pi = (2\,10)\,(4\,8)\,(2\,4)\,(6\,10)\,(1\,9)\,(3\,7)\,(1\,3)\,(9\,5)\,,$$

with overall cost equal to 40.

3) STD: It can be shown that the STDs of σ_1 and σ_2 are

$$\sigma_1 = (1\,7)\,(7\,3)\,(3\,9)\,(9\,5)\,,$$
$$\sigma_2 = (2\,8)\,(8\,4)\,(4\,10)\,(10\,6)\,,$$

each with cost 28. The total cost of the STD is $S(\pi) = 56$.

4) Merging cycles: Instead of finding the minimum cost MLD of each cycle separately, we may join the cycles and find the minimum cost MLD of a larger cycle. Here, we find the minimum cost MLD of $\sigma' = (1\,2)\,\pi = (1\,7\,3\,9\,5\,2\,8\,4\,10\,6)$. The minimum cost of an MLD of σ' can be shown to be 37. Since the cost of the transposition $(1\,2)$ must also be accounted for, the total cost is 38. Observe that this cost is smaller than the minimum MLD cost of part 2, and hence merging cycles may provide better solutions than the ones indicated by the bound (24) or as obtained through optimization of individual cycles.

□

VII. Conclusions

We introduced the problem of minimum cost transposition decomposition and presented an algorithm for computing a transposition decomposition of an arbitrary permutation, with cost at most four times the smallest possible cost. We also described an algorithm that finds the minimum cost of each transposition in terms of a product of other transpositions, as well as an algorithm that computes the minimum cost, minimum length decomposition using dynamic programing methods.

We also showed that more accurate solutions are possible for two particular families of cost functions: for metric-path costs, we derived optimal decomposition algorithms, while for extended-metric-path costs, we described a 2-approximation method.

The algorithms presented in this paper are of polynomial complexity. Finding the minimum cost of a transposition has complexity $O(n^4)$. Given the optimized cost transpositions, the minimum length decomposition can also be constructed in $O(n^4)$ steps. Computing a decomposition whose cost does not exceed the minimum cost by more than a factor of four requires $O(n^4)$ steps as well.

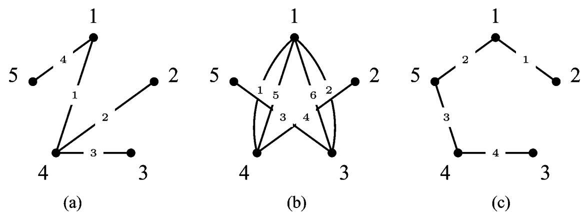

Fig. 18. Minimum cost MLD (a), MCD (b), and STD (c), for $\sigma = (12345)$. Edge labels denote the order in which transpositions are applied. (a) Min cost MLD (b) MCD (c) STD.

Appendix

We describe an algorithm for finding the path that solves

$$\arg \min_{p\in P(a,b)} \left(2\mathrm{cost}(p) - \max_{(xy)\in p} \varphi(x,y)\right), \tag{26}$$

for a, $b \in [n]$, where $P(a,b)$ is the set of all paths between a and b in $\mathcal{K}(\varphi)$. Recall from the discussion after Lemma 8 that we can obtain an MCD of (ab) from this path. The algorithm represents a variant of the Bellman-Ford procedure, described in detail in [28].

Recall that for each path $p = v_1 \cdots v_{m+1}$ in $\mathcal{K}(\varphi)$, the standard cost of the path is

$$\mathrm{cost}\,(p) = \sum_{i=1}^{m} \varphi\,(v_i, v_{i+1})\,. \tag{27}$$

Define the transposition path cost

$$\bar{\varphi}\,(p) := 2\mathrm{cost}\,(p) - \max_i \varphi\,(v_i, v_{i+1}) \tag{28}$$

to be the cost of the graph consisting of two copies of p minus one copy of its most expensive edge. For s, $u \in [n]$, $s \neq u$, let the path that minimizes the transposition path cost, among all paths from s to u, be denoted by $\hat{p}(s,u)$. The goal is to find $\hat{p}(s,u)$, for all s, $u \in [n]$, $s \neq u$.

Before describing our algorithm, we briefly review the standard Single-Source Bellman-Ford shortest path algorithm, and its relaxation technique.

Given a fixed source s, for each vertex $v \neq s$, the algorithm maintains an upper bound on the distance between s and v, denoted by $D(v)$.

"Relaxing" an edge (uv) means testing that the upper-bounds $D(u)$ and $D(v)$ satisfy the conditions

$$\begin{aligned} D(u) &\leq D(v) + w, \\ D(v) &\leq D(u) + w, \end{aligned} \tag{29}$$

where w denotes the cost of the edge (uv). If the above conditions are not satisfied, then one of the two upper-bounds can be improved, since one can reach u by passing through v, and vice versa.

In our algorithm, we maintain the upper-bound for two types of costs. The source s is an arbitrary vertex in $\mathcal{K}(\varphi)$. For a path between s and a vertex v, we use $D_1(v)$ to denote the bound on the minimum transposition path cost, and we use $D_2(v)$ to denote the bound on twice the minimum cost of the path. From the definitions of these costs, the relaxation inequalities become

$$\begin{aligned} &D_2(u) \leq 2w + D_2(v), \\ &D_2(v) \leq 2w + D_2(u), \\ &D_1(u) \leq \min\{w + D_2(v), 2w + D_1(v)\}, \\ &D_1(v) \leq \min\{w + D_2(u), 2w + D_1(u)\}. \end{aligned} \tag{30}$$

The relaxation algorithm for these inequalities, Alg. 3, is straightforward to implement.

Algorithm 3 RELAX (u, v)

```
1: $w \leftarrow \varphi(u, v)$
2: **if** $D_2(v) > D_2(u) + 2w$ **then**
3: $D_2(v) \leftarrow D_2(u) + 2w$
4: $pred_2(v) \leftarrow (u, 2)$
5: **if** $D_2(u) > D_2(v) + 2w$ **then**
6: $D_2(u) \leftarrow D_2(v) + 2w$
7: $pred_2(u) \leftarrow (v, 2)$
8: **if** $D_1(v) > D_2(u) + w$ **then**
9: $D_1(v) \leftarrow D_2(u) + w$
10: $pred_1(v) \leftarrow (u, 2)$
11: **if** $D_1(u) > D_2(v) + w$ **then**
12: $D_1(u) \leftarrow D_2(v) + w$
13: $pred(u, 1) \leftarrow (v, 2)$
14: **if** $D_1(v) > D_1(u) + 2w$ **then**
15: $D_1(v) \leftarrow D_1(u) + 2w$
16: $pred_1(v) \leftarrow (u, 1)$
17: **if** $D_1(u) > D_1(v) + 2w$ **then**
18: $D_1(u) \leftarrow D_1(v) + 2w$
19: $pred_1(u) \leftarrow (v, 1)$
```

To describe the properties of the output of the Bellman-Ford algorithm, we briefly comment on a simple property of the algorithm, termed the path-relaxation property.

Suppose $p = v_1 \cdots v_{m+1}$ is the shortest path (in terms of (27) and (28)) from $s = v_1$ to $u = v_{m+1}$. After relaxing the edges $(v_1v_2), (v_2v_3), \ldots, (v_mv_{m+1})$, in that given order, the upper-bound $D_i(u)$ (for $i = 1, 2$) equals the optimal cost of the corresponding path. Note that the property still holds even if the relaxations of the edges $(v_1v_2), (v_2v_3), \ldots, (v_mv_{m+1})$ are interleaved by relaxations of some other edges. In other words, it suffices to identify only a subsequence of relaxations of the edges $(v_1v_2), (v_2v_3), \ldots, (v_mv_{m+1})$.

In the algorithm below, we use $pred_i(v)$ to denote the predecessor of node v used for tracking the updates of the cost $D_i(v)$, $i = 1, 2$, and (u, i), $i = 1, 2$, to indicate from which of the two costs, minimized over in (30), u originated. Note that this notion of predecessor is not to be confused with the predecessor of an element in a permutation.

The modified Bellman-Ford algorithm, Alg. 4, performs $n-1$ rounds of relaxation on the edges of the graph $\mathcal{K}(\varphi)$. Lemma 26 proves the correctness of the algorithm.

Algorithm 4 SINGLE-SOURCE BELLMAN-FORD (s)

```
1: Input: vertex $s$
2: Output: $\hat{p}(s, u)$ for $1 \leq u \leq n$
3: **for** $u \leftarrow 1 \cdots n$ **do**
4: $D_1(u) \leftarrow \infty$
5: $D_2(u) \leftarrow \infty$
6: $D_1(s) \leftarrow 0$
7: $D_2(s) \leftarrow 0$
8: **for** $i \leftarrow 1 \cdots n - 1$ **do**
9: **for** each edge $(uv) \in E(\mathcal{K}(\varphi))$ **do**
10: RELAX $(u, v)$
11: **for** $u \leftarrow 1 \cdots n$ **do**
12: initialize path at $u$
13: backtrack min cost alg to recover path to $s$
14: output $\hat{p}(s, u)$
```

An example of the steps of Alg. 4 is given in Fig. 19. As a result of the relaxation of edges (ab) and (ac), neighbors of a have finite costs, as shown in Fig. 19(a). Edge (bd) is relaxed next, as seen in Fig. 19(b). Then, the relaxation of edge (cd) reduces $D_1(d)$ from 12 to 10. Continuing with the algorithm, we obtain the final result in Fig. 19(f). Note that in this example, the result obtained after the first pass is the final result. In general, however, the final costs may be obtained only after all $n - 1$ passes are performed.

Lemma 26: Given n, a cost function φ, and a source s, after the execution of Alg. 4, one has $D_1(u) = \varphi^*(s, u)$ and $D_2(u) = 2\text{cost}(p^*(s, u))$.

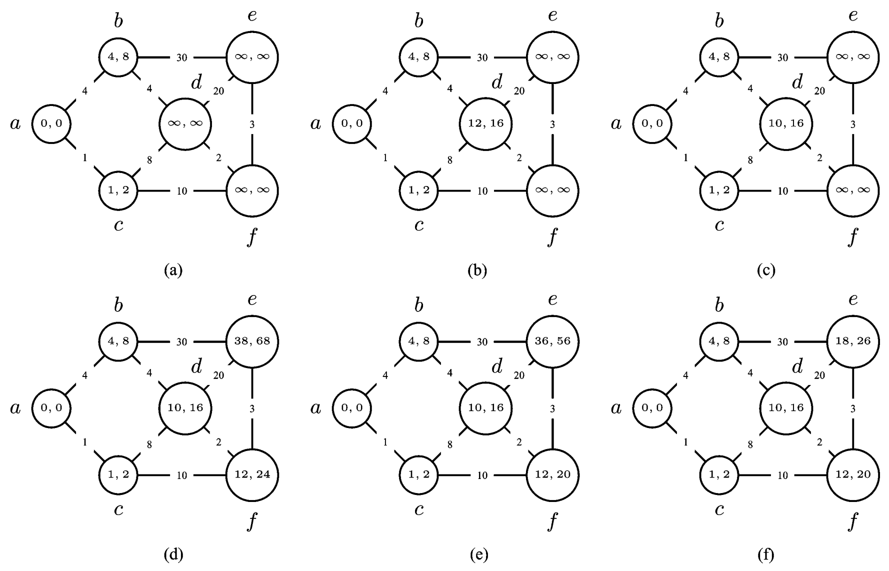
(a) a 0,0; b 4,8; c 1,2; d ∞,∞; e ∞,∞; f ∞,∞
(b) a 0,0; b 4,8; c 1,2; d 12,16; e ∞,∞; f ∞,∞
(c) a 0,0; b 4,8; c 1,2; d 10,16; e ∞,∞; f ∞,∞
(d) a 0,0; b 4,8; c 1,2; d 10,16; e 38,68; f 12,24
(e) a 0,0; b 4,8; c 1,2; d 10,16; e 36,56; f 12,20
(f) a 0,0; b 4,8; c 1,2; d 10,16; e 18,26; f 12,20
30
4
4
20
3
1
8
2
10

Fig. 19. Single-Pair Bellman-Ford algorithm on a 6-vertex graph. The costs $(D_1(u), D_2(u))$, are shown inside each vertex. Edges that are not drawn have weight ∞.(a) Relaxation of (ab) and (ac) (b) Relaxation of (bd) (c) Relaxation of (cd) (d) Relaxation of (be) and (cf) (e) Relaxation of (de) and (df) (f) Relaxation of (fe).

Proof: Let $\hat{p}(s,u) = v_1 v_2 \cdots v_{m+1}$ be the path that minimizes $\bar{\varphi}(p)$ among all paths p between $v_1 = s$ and $v_{m+1} = u$. Since any path p has at most n vertices, we have $m \leq n-1$. The algorithm makes $n-1$ passes and in each pass relaxes all edges of the graph. Thus, there exists a subsequence of relaxations that relax $(v_1 v_2), (v_2 v_3), \ldots, (v_m v_{m+1})$, in that order. The proof for the claim regarding $D_1(u)$ follows by invoking the path-relaxation property and the fact that $\varphi^*(s,u) = \bar{\varphi}(\hat{p}(s,u))$. The proof for the claim regarding $D_2(u)$ is similar. ■

Acknowledgment

The authors gratefully acknowledge useful discussions with C.-Y. Chen, C. Chekuri, and A. Orlitsky. They would also like to thank N. Kashyap for describing Example 1 and for many insightful comments and discussions.

References

[1] F. Farnoud, C.-Y. Chen, O. Milenkovic, and N. Kashyap, "A graphical model for computing the minimum cost transposition distance," in *Proc. IEEE Inf. Theory Workshop*, Dublin, Ireland, Aug./Sep. 2010.
[2] F. Farnoud and O. Milenkovic, "Decomposing permutations via cost-constrained transpositions," in *Proc. IEEE Int. Symp. Inf. Theory*, Saint Petersburg, Russia, Jul./Aug. 2011.
[3] I. P. Goulden and D. M. Jackson, *Combinatorial enumeration*. New York: Dover, 2004.
[4] J. H. van Lint and R. M. Wilson, *A Course in Combinatorics*. Cambridge, U.K.: Cambridge Univ. Press, 2001.
[5] F. Chung, "An algebraic approach to switching networks," Bell Laboratories Internal Memorandom, 1978.
[6] M. Hofri, *Analysis of Algorithms: Computational Methods and Mathematical Tools*. Oxford, U.K.: Oxford Univ. Press, 1995.
[7] M. R. Jerrum, "The complexity of finding minimum-length generator sequences," *Theor. Comput. Sci.*, vol. 36, no. 2–3, pp. 265–289, 1985.
[8] R. Weinberg, *The Biology of Cancer*. New York: Garland Science, 2006.
[9] M. von Grotthuss, M. Ashburner, and J. Ranz, "Fragile regions and not functional constraints predominate in shaping gene organization in the genus drosophila," *Genome Res.*, vol. 20, no. 8, pp. 1084–1084, 2010.
[10] M. Alekseyev and P. Pevzner, "Breakpoint graphs and ancestral genome reconstructions," *Genome Res.*, vol. 19, no. 5, pp. 943–943, 2009.
[11] M. Kothari and B. Moret, "An experimental evaluation of inversion-and transposition-based genomic distances through simulations," in *Proc. IEEE Symp. Comput. Intell. Bioinf. Comput. Biol. (CIBCB)*, Apr. 2007.
[12] D. Blackwell, *Information Theory*, ser. Modern Mathematics for the Engineer: Second Series. New York: McGraw-Hill, 1961.
[13] R. B. Ash, *Information Theory*. New York: Courier Dover, 1990.
[14] H. Permuter, P. Cuff, B. V. Roy, and T. Weissman, "Capacity of the trapdoor channel with feedback," *IEEE Trans. Inf. Theory*, vol. 54, no. 7, pp. 3150–3165, 2008.
[15] A. Jiang, M. Schwartz, and J. Bruck, "Error-correcting codes for rank modulation," in *Proc. IEEE Int. Symp. Inf. Theory*, Toronto, ON, Canada, Jul. 2008, pp. 1736–1740.
[16] A. Barg and A. Mazumdar, "Codes in permutations and error correction for rank modulation," *IEEE Trans. Inf. Theory*, vol. 56, no. 7, pp. 3158–3165, Jul. 2010.
[17] H. Chadwick and L. Kurz, "Rank permutation group codes based on Kendall's correlation statistic," *IEEE Trans. Inf. Theory*, vol. IT-15, no. 2, pp. 306–315, Mar. 1969.
[18] A. Torsi, Y. Zhao, H. Liu, T. Tanzawa, A. Goda, P. Kalavade, and K. Parat, "A program disturb model and channel leakage current study for sub-20 nm nand flash cells," *IEEE Trans. Electron Devices*, vol. 58, no. 1, pp. 11–16, 2011.
[19] N. Piratla, A. Jayasumana, A. Bare, and T. Banka, "Reorder buffer-occupancy density and its application for measurement and evaluation of packet reordering," *Comput. Commun.*, vol. 30, no. 9, pp. 1980–1993, 2007.
[20] A. El-Atawy and E. Al-Shaer, "Building covert channels over the packet reordering phenomenon," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, 2009, pp. 2186–2194.
[21] J. Bellardo and S. Savage, "Measuring packet reordering," in *Proc. 2nd ACM SIGCOMM Workshop Internet Measurment*, 2002, pp. 97–105.
[22] L. Bulteau, G. Fertin, and I. Rusu, "Sorting by transpositions is difficult," in Arxiv preprint, arXiv:1011.1157 2010.
[23] R. Y. Pinter and S. Steven, "Genomic sorting with length-weighted reversals," *Genome Inf.*, vol. 13, pp. 103–111, 2002.
[24] P. Pevzner and G. Tesler, "Transforming men into mice: The nadeau-taylor chromosomal breakage model revisited," in *Proc. 7th Annu. Int. Conf. Res. Comput. Mol. Biol.*, New York, 2003, pp. 247–256.
[25] D. B. West, *Combinatorial Mathematics*. : Preliminary version, 2009.
[26] N. Meier and J. Tappe, "Ein neuer beweis der nakayama-vermutung uber die blockstruktur symmetrischer gruppen," *Bull. London Math. Soc.*, vol. 8, no. 1, pp. 34–37, 1976.
[27] N. Kashyap, Personal communication, 2010.
[28] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA: MIT, 2001.
[29] I. Goulden, "Tree-like properties of cycle factorizations," *J. Combinatorial Theory, Series A*, vol. 98, no. 1, pp. 106–117, Apr. 2002.
[30] D. Knuth, *The Art of Computer Programming: Generating All Combinations and Partitions*. Reading, MA: Addison-Wesley, 2005.
[31] E. Slutsky, "Uber stochastische asymptoten und grenzwerte," *Metron*, vol. 5, 1925.

Farzad Farnoud (Hassanzadeh) (S'11) received his B.Sc. degree in Electrical Engineering from Sharif University of Technology, Iran, on August 2006. He received his M.Sc. degree in Electrical and Computer Engineering from University of Toronto, Canada. His Master's thesis is titled "Reliable Broadcast of Safety Messages in Vehicular Ad hoc Networks" and describes a novel method for broadcasting critical messages in a vehicular environment. He is currently a Ph.D. candidate in Electrical and Computer Engineering at UIUC. His current research interests are probability estimation of sources with large alphabets and biological sequence analysis.

Olgica Milenkovic (S'01–M'11) received the M.S. degree in mathematics and the Ph.D. degree in electrical engineering from the University of Michigan, Ann Arbor, in 2002. She is currently an associate professor at the University of Illinois, Urbana-Champaign. Her research interests include the theory of algorithms, bioinformatics, constrained coding, discrete mathematics, error-control coding, and probability theory.