Sorting permutations with a transposition tree

Bhadrachalam Chitturi\textsuperscript{1,3} and Indulekha T S\textsuperscript{2}

\textsuperscript{1} Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham, Amritapuri Campus, India
\textsuperscript{2} Department of Computer Science and Applications, Amrita Vishwa Vidyapeetham, Amritapuri Campus, India
\textsuperscript{3} Dept. of C S, University of Texas at Dallas, Richardson, TX
bhadrachalam@am.amrita.edu, indulekhats@am.amrita.edu

Abstract. The set of all permutations with \( n \) symbols is a symmetric group denoted by \( S_n \). A transposition tree, \( T \), is a spanning tree over its \( n \) vertices \( V_T = \{1, 2, 3, \ldots, n\} \). The set of permutations with \( n \) symbols forms a symmetric group denoted by \( S_n \). Let \( \pi \in S_n \) be the permutation that needs to be sorted. Let \( E_T \) be the set of edges of \( T \). Let \( \pi(i) \) and \( V_T^i \) (or simply \( i \)) denote the \( i \)th symbol of \( \pi \) and the \( i \)th vertex respectively. We employ the notation from \[ \text{[1,4,5,11]} \]. The symbol \( \pi(i) \) resides at vertex \( i \) and is called as a marker. If \( \pi(i) = j \) then position \( j \) is home for the marker \( \pi(i) \). Note that \( j \) is the rank

1 Introduction

A transposition tree \( T = (V_T, E_T) \) is a spanning tree over \( V_T \) where the cardinality of \( V_T \) is \( n \) and \( V_T = \{1, 2, \ldots, n\} \).\[ \text{[11]} \]. The set of permutations with \( n \) symbols forms a symmetric group denoted by \( S_n \). Let \( \pi \in S_n \) be the permutation that needs to be sorted. Let \( E_T \) be the set of edges of \( T \). Let \( \pi(i) \) and \( V_T^i \) (or simply \( i \)) denote the \( i \)th symbol of \( \pi \) and the \( i \)th vertex respectively. We employ the notation from \[ \text{[1,4,5,11]} \]. The symbol \( \pi(i) \) resides at vertex \( i \) and is called as a marker. If \( \pi(i) = j \) then position \( j \) is home for the marker \( \pi(i) \). Note that \( j \) is the rank
of $\pi(i)$ when $\pi$ is sorted ascending. An edge $(i, j) \in E_T$ signifies that $\pi(i)$ and $\pi(j)$ can be interchanged, i.e. swapped. A move refers to one such interchange. $d_T(\pi)$ denotes the minimum number of moves to transform $\pi$ into the identity permutation $I_n = (1, 2, 3, 4, \ldots n)$ employing the generator set $E_T$. Since there is a path between every pair of vertices in a spanning tree, any two symbols of $T$ can be swapped by a sequence of moves. Thus, $T$ generates the symmetric group $\mathcal{S}_n$.

Let $S$ be a group and $G$ be the associated set of generators. The Cayley graph $\Gamma$ of $S$ and $G$ is a graph having one vertex for each member of $S$ and an edge $(a, b) \rightarrow \exists g \in G$ such that $ag = b$. Given a transposition tree $T$, $E_T$ denotes the set of generators. A specific edge $e = (i, j) \in E_T$ is a specific generator. An upper bound is a value $k$ such that any $\pi$ in $S_n$ can be sorted in at most $k$ moves. Thus, any two vertices in $\Gamma$ are at most $k$ edges apart. The distance between $V_i$ and $V_j$ in $T$ is represented by $d_T(i, j)$. $diam(\Gamma)$, the diameter equals the maximum distance between any pair of vertices. An exact upper bound to sort any $\pi \in S_n$ equals $diam(\Gamma)$. In this article we seek to compute an upper bound for $diam(\Gamma)$.

Certain Cayley graphs, were shown [5] to have a diameter that is sub-logarithmic in the number of vertices, $n!$. For the prefix reversal operation $diam(\Gamma)$ is known to be linear; the best known upper bound is $18n/11$ [7]. For prefix transposition the best known upper bound is $n - \log_{7/2} n$ [6]. Thus, Cayley networks replaced hypercubes whose $diam(\Gamma)$ equals $\log(|V_T|)$ as the choice for interconnection networks and they have additional properties like vertex symmetry [5,12,14]. Jerrum showed that the problem of identifying a minimum length sequence of generators when the number of generators is $\geq 2$ is intractable [13].

So, in general, given some $T$ on $n$ vertices, the computation of $diam(\Gamma)$ is NP-hard and efficient computation of tight upper bound for $diam(\Gamma)$ is sought. The research in the area of Cayley graphs has been active [22,18,20,17]. Recently, cube-connected circulants topology is shown to be better than some well-known network topologies [19]. In addition to permutations, such distance measures and their upper bounds have been extensively studied on strings [9,24,23,25].

## 2 Background

Given a transposition tree $T_n = (V_T, E_T)$, $\forall e_i \in E_T$ is a generator, and the application of any one of the $n - 1$ generators is a move. For a given operation say flip (prefix reversal) on $\pi \in S_n$, a prefix of length $k$ where $2 \leq k \leq n$ is reversed corresponding to $n - 1$ generators $k = 2 \ldots n$.

Determining good upper bounds for sorting permutations under various operations is of interest. The computation of exact upper bound for sorting permutations with many operations is either intractable or its complexity is unknown [9]. We state some results in the general area of sorting permutations with various generator sets. In 2009 Chitturi et al. [7] improved the upper bound given by Gates and Papadimitriou [10] for sorting permutations with prefix reversals.
The problem of computing the diameter of the Cayley graph generated by cyclic adjacent transpositions was introduced by Jerrum \[13\]; for which Feng et al. \[8\] prove a lower bound of \(n^2/4\). An \(O(n^3)\) amortized time algorithm to compute the optimum number of moves to sort any permutation with transposition operation was designed \[22\].

Akers and Krishnamurthy computed an upper bound \(f(\Gamma)\) for the \(\text{diam}(\Gamma)\) for transposition trees in \(\Omega(n!n^2)\) time \[6\]. Given a transposition tree \(T\), Ganesan \[4\] computes a non-deterministic measure \(\beta\), an estimate of the exact upper bound. \(\beta_{\text{max}}\) is the maximum among all values of \(\beta\). Only \(\beta_{\text{max}}\) that requires exponential time to compute is an upper bound. It is shown that \(\beta_{\text{max}} \leq f(\Gamma)\).

Kraft \[3\] proposes three algorithms to identify upper bounds on \(\text{diam}(\Gamma)\) generated using transposition trees: \(\alpha\) an exponential time algorithm, \(\eta\) and \(\xi\) a randomized algorithm. The method \(\alpha\) tries to improve the bound of \[5\] by identifying the minimal value at each step.

The terminology used in \[16,12,11\] is adopted here. Let \(T\) be transposition tree on a vertex set \(\{1, 2, \ldots, n\}\). Let \(\pi\) be the permutation in \(S_n\) that is to be sorted. If \(\pi = (7, 6, 5, 4, 3, 2, 1)\) then \(\pi \in S_7\) in cycle representation \(\pi = (1, 7)(2, 6)(3, 5)(4)\) that is, \(\pi\) has four cycles i.e. \(\eta(\pi) = 4\) \[5\]. Likewise, if \(\pi = (6, 5, 4, 3, 2, 1) = (1, 6)(2, 5)(3, 4)\) then \(\eta(\pi) = 3\). A marker \(\pi(i)\) resides at position \(i\). The move which swaps elements at positions \(i\) and \(j\) is denoted as \((i \ j)\). If \((i \ j)\) is executed on \(\pi\) then we obtain \(\pi(i \ j)\), the result of application of \((i \ j)\) to \(\pi\). Application of a sequence of generators so that \(\pi(j)\) reaches its home is called homing \(\pi(j)\).

The diameter of the Cayley graph is identified only for some transposition trees. If \(T\) is \(K_{1,n-1}\), a star graph, the diameter is \(\lceil (n-1)/2 \rceil\) \[5\] for bubble sort graph, it equals \(nC_2\). Recently, \[2\] identified diameter for two novel classes of trees \(S_{m,k}\) and \(M_k\). \(S_{m,k}\) has one central vertex \(\text{center}\) and \(m\) spokes, each spoke is a path of length \(k+1\) where all the paths share \(\text{center}\). Thus, the regular star tree with \(k-1\) leaves i.e. \(S_k\) or \(S_{1,k-1}\) is same as \(S_{k-1,1}\). Let \(f(S_{m,k})\) be the diameter of the Cayley graph generated by \(S_{m,k}\) then \(f(S_{m,k}) = mk(2k + 1)/2\) \[2\]. Matchstick tree \(M_k\) was defined in \[21\]. It is an extension of the path graph with \(k\) vertices where each vertex in the path graph has a corresponding leaf attached to it. The diameter of a Cayley graph generated with \(M_k\) is \(k^2 + k - 1\) for \(k > 2\) \[2\).

One can adopt brute force search to compute the diameter corresponding to \(T_n\), for \(n < 6\). In the current article \(n \geq 6\). In Section 3 we show the computation of \(\delta^*\) in \(O(n^2)\) time. In Section 4 we derive expression for \(\delta^*\) and compare it to the best existing bounds.

### 3 Algorithm \(\delta^*\)

Eccentricity of a vertex \(u\) in \(V_T\), denoted by \(\text{ecc}(u)\), equals the maximum value of \(d_T(u, i)\) for all \(i\). The center of a tree is either a vertex (centered tree) or an edge (bicentred tree). A unique vertex with minimum eccentricity forms the center of a tree. An adjacent pair of vertices (corr. to an edge) with minimum eccentricity
form the center of a bicentered tree. Let \( \mu \) be the upper bound computed by an algorithm \( A \) on a given tree \( T \). \( \text{Cum}_{\mu}(n) \) is the cumulative value of upper bounds of all trees with \( n \) vertices computed by \( A \). Likewise \( \text{Cum}_{\mu}(X) \) is the cumulative value of upper bounds computed by \( A \) for all trees belonging to class \( X \).

Chitturi designed an algorithm \( \text{Algorithm } S \) that identifies the set of all vertices that have maximum eccentricity i.e. \( S \) in \( T \) in linear time \([1]\). The general idea of the algorithms in \([1]\) is to delete a set of leaves say \( X \) and obtain an upper bound on the number of moves that suffice to home markers to all vertices in \( X \). A cluster \( C \) in \( S \) is the maximal subset of \( S \) such that any \( u, v \) in \( C \) are less than \( \text{diam}(T) \) apart. Note that if all but one of the clusters is deleted then the diameter of the resultant tree decreases. Based on this idea, Algorithm \( D' \) computes an upper bound \( \delta' \) that works by deleting all clusters in \( S \) except the largest cluster \( C^* \) \([1]\). It employs a linear time algorithm \( \text{Fast}_{NC} \) to identify all clusters \([1]\). Algorithm \( D' \) has two versions, \( D'_{v_1} \) and \( D'_{v_2} \) which calculates \( \delta'_{v_1} \) and \( \delta'_{v_2} \) respectively. \( D'_{v_1} \) removes the entire \( S \) if \( |S - C^*| > (2/3)|S| \) while \( D'_{v_2} \) removes the entire \( S \) if \( |S - C^*| = (2/3)|S| \). It was shown that \( D'_{v_1} \) algorithm has the best value for \( \text{Cum}_{\delta'}(n) \) and \( \text{Cum}_{\delta'}(B) \) where \( B \) denotes full binary trees. Here we improve upon Algorithm \( D' \) and design a new algorithm \( \delta^* \) that computes the new upper bound \( \delta^* \). Results show that \( \text{Cum}_{\delta^*}(n) > \text{Cum}_{\delta'}(n) \) for \( n \in 6 \ldots 15 \). The following observations form the basis of Algorithm \( \delta^* \).

**Observation 1:** If \( |S \setminus C^*| > |C^*| \) then at most \( |C^*| \) markers need \( \text{diam}(T) \) moves each to be homed.

**Proof:** The proof follows from pigeon hole principle. Let the set of vertices that are being deleted be \( X \). If \( |X| > |C^*| \) then at most \( |C^*| \) from \( C^* \) can be homed to the vertices of \( X \) the rest must be from elsewhere including within \( X \).

**Theorem 1** \( \delta^* \) is an upper bound for sorting permutations.

**Proof:** The set of vertices that is to be deleted, i.e. \( S \setminus C \) where \( C \) is one of the largest clusters be called by \( X \). \( X \) can be union of several clusters and \( X \subset S \) where \( S \) is the set of vertices in \( T \) with greatest eccentricity \([1]\). Recall that deletion implies that markers are being homed to the vertices in \( X \). We try to obtain an upper bound on the cost to home all markers to \( X \). The scenario yields Case 1 and Case 2. The other scenarios clearly yield a lower value as the markers are less that \( \text{diam}(t) \) apart from their respective homes (destinations). Case 1): All markers homed to vertices in \( X \) are from \( X \). Case 2): Markers homed to vertices in \( X \) are from any vertices of \( S \).

Case 1): Lemma 5 of \([1]\) gives an upper bound of \( |X| \) \( (\text{diam}(T) - 1/2) \) for Case 1. An additional half move is subtracted when \( |X| \) is odd.

Case 2): Let \( |X| \) be \( x \) and let \( |C| \) be \( c \). Let \( i \) vertices from \( C \) be homed to corresponding \( i \) vertices in \( X \). The associated cost is \( i \text{diam}(T) \) moves. It follows that \( x - i \) vertices from \( X \) are homed among themselves where the upper bound for the associated cost is \( (x - i)(\text{diam}(T) - 1/2) \) due to Lemma 5 of \([1]\). When
Algorithm 1 Algorithm $\delta^*$

1: $T$ and $V_T$ are the current tree and the corr. set of vertices.
2: \textbf{S1.}
3: $\delta^* \leftarrow 0$
4: if $T$ is a star graph then
5: \hspace{1em} $\delta^* \leftarrow \delta^* + \lfloor 3/2(|V_T| - 1) \rfloor$ and terminate.
6: \hspace{1em} end if
7: \textbf{S2.}
8: Identify $S$ by executing \textit{Algorithm S}.
9: \textbf{S3.}
10: Compute clusters for $S$ with \textit{Algorithm NC}.
11: \textbf{S4.}
12: Identify $C = S - C^*$ where $C^*$ is the largest cluster and $\text{distSum}(C^*)$ is the least (break ties arbitrarily). $V_C$: vertices of $C$. $\text{diam}(T \setminus V_C) < \text{diam}(T)$.
13: \textbf{Case 1:} $|C^*| \geq |S|/2$
14: \hspace{1em} $\delta^* \leftarrow \delta^* + |C| \ast \text{diam}(T)$; $T \leftarrow T \setminus V_C$
15: \textbf{End Case 1}
16: \textbf{Case 2:} $|C^*| < |S|/2$ \hspace{0.5em} $\lceil |C| \geq |C^*| \rceil$
17: \hspace{1em} $\delta^* \leftarrow \delta^* + |C^*| \ast \text{diam}(T)$; $T \leftarrow T \setminus V_C$
18: if $|C| - |C^*|$ is even then
19: \hspace{2em} $\delta^* \leftarrow \delta^* + (|C| - |C^*|) \ast (\text{diam}(T) - 1/2)$;
20: \textbf{else}
21: \hspace{2em} $\delta^* \leftarrow \delta^* + (|C| - |C^*|) \ast (\text{diam}(T) - 1/2) - 1/2$;
22: \textbf{end if}
23: \textbf{End Case 2}
24: if $T$ is not a star graph then go to step S2.
25: \textbf{else}
26: \hspace{1em} $\delta^* \leftarrow \delta^* + \lfloor 3/2(|V_T| - 1) \rfloor$ and terminate.
27: \textbf{end if}

$x - i$ is odd then cost reduces by 1/2. Thus, the total cost is $(x - i)(\text{diam}(T) - 1/2) + i \cdot \text{diam}(T)$ which maximizes when $i$ is maximized. That is, $i = c$ yielding $(x - c)(\text{diam}(T) - 1/2) + c \cdot \text{diam}(T)$. When $x - c$ is odd then cost reduces by 1/2. Further, consider the scenario where $u \in C$ is to be homed to $v \in X$ and $v \in X$ is to be homed to $w \in X$. Note that homing $u$ first moves $v$ one edge closer to its home. So, it requires at most $\text{diam}(T) - 1$ moves. Thus, this scenario does not yield the worst case. Further, if there are dependencies such as the home of $a$ is $b$, the home of $b$ is $c$ etc. then the dependency that yields the worst case is shown to be mutual swap of pairs of markers; that is, the home of $a$ is $b$ and the home of $b$ is $a$ (Lemma 5 of [1]). So, given a subset $Y$ of $S$ where the vertices of $Y$ are to be homed within $Y$, the maximum number of pairs of $Y$ are swapped employing their corresponding sequence of moves. Thus, the theorem follows.

4 Results

The cumulative sum of all upper bound values for all trees with up to 10 and 15 vertices was recorded for all the existing algorithms in [1] and [2] respectively.
yielded the minimum value \[ \deltav^* \]. Our results show that \( \deltav^* \) yields a smaller value than \( \deltav' \) for the same. We obtained all non-isomorphic trees with a given number of vertices from sagemath.org [15]. The results are tabulated in Table 1. We theoretically show that for a full binary tree \( \deltav^* \) indeed is tighter than \( \deltav' \). In [1] \( \deltav' \) is shown to be deterministic (the choices of deletion do not alter the value of the measure). We show that \( \deltav^* \) is also deterministic by proving that various sequences of deletions of vertices leads to isomorphic trees. So, the comparison is valid. Table 2 shows the corresponding execution results.

**Table 1.** Comparison of \( \deltav^* \) with other methods for all trees with \( n \) vertices.

| No. of nodes | \( \text{Cum}_{\deltav'} \) | \( \text{Cum}_{\deltav^*} \) | \( \text{Cum}_{\deltav^*} \) |
|--------------|-----------------|-----------------|-----------------|
| 6            | 63              | 63              | 63              |
| 7            | 154             | 153             | 153             |
| 8            | 409             | 407             | 407             |
| 9            | 1032            | 1028            | 1027            |
| 10           | 2819            | 2809            | 2805            |
| 11           | 7401            | 7376            | 7361            |
| 12           | 20277           | 20222           | 20175           |
| 13           | 50032           | 49931           | 49820           |
| 14           | 152585          | 152285          | 151855          |
| 15           | 212841          | 212532          | 212217          |

**Table 2.** Comparison of \( \deltav^* \) with \( \deltav' \) on a full binary tree with depth \( d \).

| No. of nodes | No. of leaves | Depth | \( \deltav' \) | \( \deltav' \) | \( \deltav^* \) |
|--------------|---------------|-------|---------------|---------------|---------------|
| 3            | 2             | 1     | 3             | 3             | 3             |
| 7            | 4             | 2     | 17            | 17            | 15            |
| 15           | 8             | 3     | 58            | 58            | 55            |
| 31           | 16            | 4     | 171           | 172           | 167           |
| 63           | 32            | 5     | 460           | 461           | 453           |
| 127          | 64            | 6     | 1165          | 1168          | 1153          |
| 255          | 128           | 7     | 2830          | 2833          | 2807          |

Algorithm \( \deltav^* \) ensures that in every iteration, the maximum cost required to home a node is determined by the availability of room for homing that node. This is the qualitative improvement of this article. Let \( B(d) \) be a full binary tree of depth \( d \) with levels \((1, 2, 3, \ldots, d + 1)\) where root is at level 1. Let \( m \) be the number of leaf nodes in \( B(d) \). When the algorithm is run on \( B(d) \) for \((2d - 2)\) iterations then \( B(d) \) will be transformed into a star graph (whose centre node has degree 3, i.e. \( K_{1,3} \)). In the first iteration, the set \( S \) contains all leaf nodes.
Sorting permutations with a transposition tree

and forms two clusters. One cluster contains all the leaf nodes of the left subtree and the other contains all leaf nodes in the right subtree. Since the distance sums of both clusters are same, any one of the two clusters can be considered as the largest cluster and the other can be deleted. Let the algorithm remove \( m/2 \) nodes of the right cluster each at a cost of \( 2d \) (\( 2d \) is the initial diameter). If the left cluster is chosen for deletion then the resultant tree is isomorphic to the resultant tree in the previous case and the cost is identical. In the second iteration, two clusters are formed. One cluster \( C_1 \) with \( m/2 \) leaf nodes of left subtree and another \( C_2 \) with \( m/4 \) leaf nodes of right subtree. The smaller of the clusters, i.e. \( C_2 \) is deleted. So, in the second iteration \( m/4 \) nodes are removed each at a cost of \( 2d - 1 \) (after the first iteration, the diameter is reduced by 1).

After two iterations, nodes in the levels \( d + 1 \) and \( d \) of the right subtree of \( B(d) \) are removed. Let us call the new tree as \( T(d) \). So, \( \delta^* = A + B + C \) where,

\[
A = \text{Cost to convert } B(d) \text{ to } T(d) \text{ (first 2 iterations)}
\]

\[
B = \text{Total cost for the next } 2d - 4 \text{ iterations which results in } K_{1,3}
\]

\[
C = \delta^*(K_{1,3})
\]

A: The total cost to convert \( B(d) \) to \( T(d) \) = \( (m/2)2d + (m/4)(2d - 1) \) = \( m(6d - 1)/4 \).

B: It takes \( (2d - 4) \) iterations, i.e. \( (3 \ldots (2d - 2)) \) to transform \( T(d) \) to \( K_{1,3} \). It consists of \( (d - 2) \) odd iterations and \( (d - 2) \) even iterations. In every odd iteration, the current tree \( T \) with the current center \( c \) can be described as follows.

There is an edge from \( c \) to the center of the tree of the previous iteration that we call as upward edge. If we imagine that the subtree connected through upward edge does not exist then we obtain a binary tree. Our terminology of left and right subtree is based on such presumption. Three clusters \( C_1, C_2 \) and \( C_3 \) are formed where \( C_1 \) and \( C_2 \) are the left and right subtrees of the current tree and \( C_3 \) is the subtree that is connected to the center through upward edge. In the beginning of an odd iteration, the centre is a single node (since the diameter is even). Both \( C_1 \) and \( C_2 \) contain same number of nodes and are of maximum size, any one of these clusters (along with the third cluster \( C_3 \)) will be chosen for deletion. Let us assume that \( C_2 \) and \( C_3 \) are chosen for deletion. The nodes in \( C_2 \) will be deleted at a cost of current diameter per node and the nodes of \( C_3 \) will be deleted at a cost of current diameter \(- 1/2\) (because \( |C_1| - |C_2| = 0 \)). The trees that are obtained either by deleting \( C_1 \) and \( C_3 \) or by deleting \( C_2 \) and \( C_3 \) are isomorphic. It is shown in Figure 1.

After the \( d^{th} \) iteration, the diameter of the tree will be reduced to \( d \) so that the entire right subtree is deleted (or the left, depends on the choice of the clusters, when there is a tie). In every odd iteration after this, all the three clusters will contain same number of nodes. Irrespective of the clusters chosen for deletion, the algorithm will generate the same (isomorphic) tree. Refer Figures 2 to 4. Every even iteration will generate exactly two clusters \( C_1 \) and \( C_2 \) with different sizes. Let us assume that \( C_2 \) is the smallest cluster and hence the one chosen for deletion.

Table 3 and Table 4 show the number of nodes removed and the removal cost per node in various iterations of phase B. The removed nodes are categorized.
Fig. 1. Removal of left or right cluster

Fig. 2. First possibility

Fig. 3. Second possibility

Fig. 4. Third possibility
into two groups based on the removal cost per node.

### Table 3. Removal cost per node in the first \( d \) iterations

| Iteration | \( |C_2| \) | Cost per node in \( C_2 \) | \( |C_3| \) | Cost per node in \( C_3 \) |
|-----------|-------------|-----------------|-------------|-----------------|
| 3         | \( m/2^2 \) | 2d-2            | \( m/2^3 \) | 2d - 2 - 1/2    |
| 4         | \( 3m/2^2 \) | 2d-3            | -           | -               |
| 5         | \( m/2^3 \) | 2d-4            | \( 3m/2^5 \) | 2d - 4 - 1/2    |
| 6         | \( 7m/2^6 \) | 2d-5            | -           | -               |
| 7         | \( m/2^4 \) | 2d-6            | \( 7m/2^7 \) | 2d - 6 - 1/2    |
| 8         | \( 15m/2^8 \) | 2d-7            | -           | -               |
| ...       | ...         | ...             | ...         | ...             |
| d         | \( (2^{d/2} - 1)m/2^d \) | \( (d + 1) \) | -           | -               |
| d         | \( m/2^{(d+1)/2} \) | \( (d + 1) \) | \( (2^{d/2} - 1)m/2^d \) | \( d - 1/2 \) (for odd \( d \)) |

### Table 4. Removal cost per node in the next \( d - 2 \) iterations

| Iteration | \( |C_2| \) | Cost per node in \( C_2 \) | \( |C_3| \) | Cost per node in \( C_3 \) |
|-----------|-------------|-----------------|-------------|-----------------|
| ...       | ...         | ...             | ...         | ...             |
| \( (2d - 9) \) | 16         | 10              | 16          | 10-1/2          |
| \( (2d - 8) \) | 16         | 9               | -           | -               |
| \( (2d - 7) \) | 8          | 8               | 8           | 8-1/2           |
| \( (2d - 6) \) | 8          | 7               | -           | -               |
| \( (2d - 5) \) | 4          | 6               | 4           | 6-1/2           |
| \( (2d - 4) \) | 4          | 5               | -           | -               |
| \( (2d - 3) \) | 2          | 4               | 2           | 4-1/2           |
| \( (2d - 2) \) | 2          | 3               | -           | -               |

Thus, the total cost \( B \) is the sum of three series \( P \), \( Q \), and \( R \), where

\[
P = \frac{m(2d - 2)}{2^2} + \frac{m(2d - 4)}{2^4} + \frac{m(2d - 6)}{2^6} + \ldots + \frac{m(2d - (2d - 4))}{2^{2d-1}}
\]

\[
Q = \frac{m(2d - 2 - 1/2)}{2^3} + \frac{3m(2d - 4 - 1/2)}{2^5} + \frac{7m(2d - 6 - 1/2)}{2^7} + \ldots + \frac{(2^{d-2} - 1)m(4 - 1/2)}{2^{2d-3}}
\]

\[
R = \frac{3m(2d - 3)}{2^4} + \frac{7m(2d - 5)}{2^6} + \frac{15m(2d - 7)}{2^8} + \ldots + \frac{2^{d-2} - 1m(3)}{2^{2d-2}}
\]

\( C \): The cost required for the final star graph is \( \left\lfloor \frac{3}{2}(4 - 1) \right\rfloor \) which is 4.
4.1 Comparing $\delta^*$ with $\delta'v_1$ and $\delta'v_2$

The improvement occurs when there are more number of nodes to be removed than that of the nodes in the largest cluster chosen. In every iteration, the largest cluster chosen by both $\delta'v_1$ and $\delta^*$ contain the same set of nodes. Let $C$ be the largest cluster chosen. Since $|C| \leq \frac{1}{2}|S| < \frac{2}{3}|S|$, both $\delta'v_1$ and $\delta^*$ will delete the set $S - C$ and generate the same trees (isomorphic) after every iteration. $\delta'v_1$ deletes each node at a cost of current diameter, while $\delta^*$ deletes some of them in lesser cost. In every odd iteration in phase B, there is an improvement of 0.5 per node in the second cluster deleted. The improvement of $\delta^*$ over $\delta'v_1$ is as follows:

$$\delta^* = \delta'v_1 - \frac{(m + 1)}{6} - 1 - \frac{1}{2} \quad /\text{if } d \text{ is odd}$$
$$\delta^* = \delta'v_1 - \frac{(m + 2)}{6} - 1 \quad /\text{if } d \text{ is even}$$

**Table 5. Improvement of $\delta^*$ over $\delta'v_1$**

| Iteration | Improvement |
|-----------|-------------|
| 3         | $m/2^0$     |
| 5         | $3m/2^6$    |
| 7         | $7m/2^8$    |
| ...       | ...         |
| $(2d - 9)$| 8           |
| $(2d - 7)$| 4           |
| $(2d - 5)$| 2           |
| $(2d - 3)$| 1           |

The behaviour of $\delta'v_2$ will be different from $\delta'v_1$, when $|C^*| \leq \frac{1}{2}|S - C^*|$. In a full binary tree, from $(d + 2)^{nd}$ or $(d + 1)^{st}$ iteration onwards, (for odd and even depth trees respectively) the above condition holds. From that iteration onwards, the nodes removed by $i^{th}$ and $(i + 1)^{th}$ iteration of $\delta'v_1$ will be removed by $\delta'v_2$ in a single iteration with lesser cost. This improvement is same as that of $\delta^*$ in those iterations.

Recall that either 3 or 2 clusters will be formed in every odd and even iteration respectively. Let the clusters in the $i^{th}$ odd iteration be $C_{1,i}, C_{2,i}$, and $C_{3,i}$, and that in the $(i + 1)^{th}$ even iteration be $C_{1,i+1}$ and $C_{2,i+1}$. As discussed before, from this $i^{th}$ iteration onwards, all the three clusters formed are of same size. In the $i^{th}$ iteration, $S = |C_{1,i}| + |C_{2,i}| + |C_{3,i}|$. Any of the clusters can be chosen as $C^*$. Then, $|C| = 2|S|/3$. So, $\delta'v_2$ will delete the entire $S$, with a cost of $(current diameter - 1/2)$ per node. Let $k$ denotes the current diameter. Then, the cost in $i^{th}$ iteration of $\delta'v_2$ is $cost_{i,\delta'v_1} = (k - 1/2)(|C_{1,i}| + |C_{2,i}| + |C_{3,i}|) = 3|C_{1,i}|(k - 1/2)$. 

In the corresponding iteration, $\delta^*$ will remove only two clusters, say $C_{2,i}$ and $C_{3,i}$. So, the cost in this iteration is, $\text{cost}_{i,\delta^*} = k(|C_{2,i}|) + (k - 1/2)(|C_{3,i}|)$. In the next iteration of $\delta^*$, it will form two clusters of equal size, say $C_{1,i+1},C_{2,i+1}$. Since $C_{1,i}$ and $C_{1,i+1}$ contains the same set of nodes, removal of $C_{1,i+1}$ in this iteration will result the same tree as that in the $i^{th}$ iteration of $\delta'_{v_1}$. The cost in this iteration is $\text{cost}_{i+1,\delta^*} = (k - 1/2)(|C_{1,i}|)$. So, the total cost of $\delta^*$ in $i^{th}$ and $(i+1)^{th}$ iteration is $3|C_{1,i}|(k-1/2)$, which is same as that of $\delta'_{v_2}$ in $i^{th}$ iteration.

The final expressions obtained for each of these measures for $B(d)$: a full binary tree of depth $d$ are shown below.

\[
\delta^* = \delta'_{v_2} - \frac{(m+1)}{6} - \sqrt{\frac{m}{8}} - \frac{1}{2} \quad \text{if } d \text{ is odd}
\]
\[
\delta^* = \delta'_{v_2} - \frac{(m+2)}{6} - \sqrt{\frac{m}{4}} \quad \text{if } d \text{ is even}
\]
\[
\delta^* = 4md - 6m + O(d^2) - \frac{(m+4)}{6} \quad \text{if } d \text{ is odd}
\]
\[
\delta^* = 4md - 6m + O(d^2) - \frac{(m-4)}{6} \quad \text{if } d \text{ is even}
\]
\[
\delta'_{v_1} = 4md - 6m + O(d^2);
\]
\[
\delta'_{v_2} = 4md - 6m + O(d^2) + \sqrt{\frac{m}{8}}; \quad \text{if } d \text{ is odd}
\]
\[
\delta'_{v_2} = 4md - 6m + O(d^2) + 1 + \sqrt{\frac{m}{4}}; \quad \text{if } d \text{ is even}
\]

The above expressions show that the gain for $\delta^*$ over the other two algorithms for a $B(d)$ is approximately $m/6$ where $m$ is the number of leaf nodes and the total number of nodes is $n$. However, $m = (n+1)/2$. Thus, gain is approximately $n/12$. The theorem stated below immediately follows.

**Theorem 2** $\delta'_{v_1} - \delta^* = \Omega(n)$ for a full binary tree.

### 5 Conclusion

We design a new measure $\delta^*$. It is shown to yield the smallest cumulative sum for all trees of a given cardinality where $|V_T| \leq 15$. This is an improvement over the previously known best upper bound $\delta'$. For a full binary tree we show that $\delta^*$ is tighter than $\delta'$. In addition to full binary trees, were able to theoretically show that $\delta^*$ is tighter than $\delta'$ for several other tree classes.

### 6 Acknowledgements

Authors thank Amma for the direction.
References

1. B. Chitturi. Upper bounds for sorting permutations with a transposition tree. *Discrete Mathematics, Algorithms and Applications*, 5(01):1350003, 2013.
2. S. Uthan, and C. Chitturi. Bounding the diameter of cayley graphs generated by specific transposition trees. International Conference on Advances in Computing, Communications and Informatics (ICACCI), IEEE, 1242–1248, 2017.
3. Benjamin Kraft. Diameter of Cayley graphs generated by Transposition Tree. *Discrete Applied Mathematics*, 184:178–188, 2015.
4. A. Ganesan. An efficient algorithm for the diameter of Cayley graphs generated by transposition trees. IAENG Int. J. of Applied Mathematics, 42(4):214-223.
5. S. B. Akers and B. Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4):555–566, 1989.
6. B. Chitturi. Tighter upper bound for sorting permutations with prefix transpositions. *Theoretical Computer Science*, 602:22–31, 2015.
7. B. Chitturi, W. Fahle, Z. Meng, L. Morales, C. O. Sheilds, I. H. Sudborough and W. Voit. An (18/11)n upper bound for sorting by prefix reversals. *Theoretical Computer Science*, 410(36):3372-3390, 2009.
8. X. Feng, B. Chitturi, and H. Sudborough. Sorting Circular Permutations by Bounded Transpositions. *AEMB*, Springer-Verlag, 680(7):725–736.
9. G. Fertin, A. Labarre, I. Rusu, E. Tannier and S. Vialette. *Combinatorics of Genome Rearrangements*. The MIT Press 2009.
10. W. H. Gates and C. H. Papadimitriou. Bounds for sorting by prefix reversal. *Discrete Mathematics*, 27(1):47–57, 1979.
11. M. C. Heydemann, N. Marlin and S. Prennes. Complete Rotations in Cayley Graphs. *Eur. J. Comb.*, 22(2): 179–196, 2001.
12. M. C. Heydemann. Cayley graphs and interconnection networks. In *Graph Symmetry: Algebraic Methods and Applications*, (1997):167-226. Kluwer Academic Publishers, Dordrecht.
13. M. Jerrum. The complexity of finding minimum length generator sequences. *Theoretical Computer Science*, 36:265–289, 1985.
14. S. Lakshmivarahan, J-S. Jho and S. K. Dhall. Symmetry in interconnection networks based on Cayley graphs of permutation groups: A survey. *Parallel Computing*, 19:361–407, 1993.
15. [http://www.sagemath.org/](http://www.sagemath.org/)
16. J. H. Smith. Factoring, into Edge Transpositions of a Tree, Permutations Fixing a Terminal Vertex. *Journal of Combinatorial Theory*, 85(A):92–95, 1999.
17. B. Chitturi. Adjacencies in Permutations. *arXiv preprint arXiv:1601.04469*, 2016.
18. Dohan Kim. Sorting on graphs by adjacent swaps using permutation groups. *Computer Science Review*, 22:89–105, 2016.
19. Hamid Mokhtar. A few families of Cayley graphs and their efficiency as communication networks. *PhD. Thesis*, Dept of Math and Stats, U of Melbourne, 2016.
20. Elena Konstantinova, Alexey Medvedev. Independent Even Cycles in the Pancake Graph and Greedy Prefix-Reversal Gray Codes. *Graphs and Combinatorics*, 32(5):1965–1978, 2016.
21. B. Chitturi, D. Bein, N. V. Grishin. Complete enumeration of compact structural motifs in proteins. Proceedings of the International Symposium on Biocomputing, ACM, 19–27, 2010.
22. B. Chitturi, P. Das. Sorting permutations with transpositions in $O(n^3)$ amortized time. *Theoretical Computer Science*, 2018, 10.1016/j.tcs.2018.09.015.
23. B. Chitturi, H. Sudborough, W. Voit and X. Feng. Adjacent Swaps on Strings. *LNCS*, 5092:299–308. COCOON ’08 Proceedings of the 14th annual international conference on Computing and Combinatorics.

24. B. Chitturi. On transforming sequences. *Ph.D. Thesis*, Department of Computer Science, University of Texas at Dallas, 2007.

25. B. Chitturi. A note on Complexity of Genetic Mutations. *Discrete Mathematics Algorithms and Applications*, 3(3):269–286, 2011.