DiVA – Digitala Vetenskapliga Arkivet http://umu.diva-portal.org

---

This is an author produced version of a paper presented at **Prague Stringology Conference, August 29-31 2011, Prague.**

This paper has been peer-reviewed but does not include the final publisher proof-corrections or journal pagination.

Citation for the published paper:

Martin Berglund

Analyzing Edit Distance on Trees: Tree Swap Distance is Intractable

Proceedings of the Prague Stringology Conference 2011, 2011, p. 59-73

Analyzing Edit Distance on Trees: Tree Swap Distance is Intractable

Martin Berglund

Department of Computing Science, Umeå University
90187 Umeå, Sweden
mbe@cs.umu.se

Abstract. The string correction problem looks at minimal ways to modify one string into another using fixed operations, such as for example inserting a symbol, deleting a symbol and interchanging the positions of two symbols (a "swap"). This has been generalized to trees in various ways, but unfortunately having operations to insert/delete nodes in the tree *and* operations that move subtrees, such as a "swap" of adjacent subtrees, makes the correction problem for trees intractable. In this paper we investigate what happens when we have a tree edit distance problem with *only* swaps. We call this problem tree swap distance, and go on to prove that this correction problem is NP-complete. This suggests that the swap operation is fundamentally problematic in the tree case, and other subtree movement models should be studied.

1 Introduction

String edit distance is an old, well-known and thoroughly studied concept, most commonly used in the context of *string correction problems*. An edit distance (of which there are many kinds) defines some small set of operations on strings. An instance of the string correction problem corresponding to a given edit distance is a question of the form "can the string s be transformed into s' by applying at most k edit operations?" In more complex cases the string correction problem may associate different costs to the edit operations, having k serve as a total budget.

One of the most frequently used types of edit distance is Levenshtein distance [7], which features the three operations `delete`, `insert`, and `replace`. These can be applied to any position in a string, to delete a single symbol, insert a single symbol, and replace a single symbol by another, respectively. A popularly applied extension, called Damerau-Levenshtein distance [3], adds a fourth operation, `swap`, which swaps the position of any two symbols in a string. For both of these distances the string correction problem is very efficiently solvable if all operations have the same cost. A more general variant is called the *extended string-to-string correction problem*, which uses the four Damerau-Levenshtein operations, but allows the problem instance to assign each operator an arbitrary integer cost [11]. In general this makes the correction problem strongly NP-complete [10], a fact that we will make use of later.

As this area is well-explored and successful in the string case it is of great interest to extend the same ideas to the tree case [8, 9]. This work has been very successful for the "insert", "delete" and "replace" operations, but the "swap" operation has most often been left out [12, 5, 2]. This is in fact a necessity, as the problem quickly becomes intractable when subtree movement is introduced as an operation. This follows trivially from the fact that tree edit distance on unordered trees is NP-complete [13], by duplicating nodes one can create a situation where the swaps are so much cheaper than

a `delete/insert` operation that the problem becomes equivalent to the unordered one. Still, swaps and other subtree movement operations remain very interesting in practice in very diverse fields such as XML processing, computational biology, natural language processing and many others. Approximations have been considered, for example [1] introduces swaps into tree edit distance but the algorithm as given actually restricts each node to participate in at most one swap, so arbitrary reorderings are not possible.

While much work has been done to restrict the swaps to make the problem tractable we will here instead take a step back and consider the "tree swap distance" problem. In this restriction of tree edit distance *only* the `swap` operation is allowed, reducing the problem to finding the least number of swaps necessary to reorder one tree into another. Unfortunately the end result is that we demonstrate that even this problem is NP-complete, suggesting that the `swap` operation may be a computationally bad choice to model subtree movement operations.

2 Preliminaries

Let $\mathbb{N}$ denote the set of natural numbers $\{0, 1, 2, 3, \ldots\}$. For all $n \in \mathbb{N}$ let $[n]$ denote the set $\{1, \ldots, n\}$. An *alphabet* Σ is a finite set of symbols. Going forward we will simply use Σ to mean some appropriate alphabet without specifying it precisely. The empty string/sequence is denoted by ϵ. The set of all strings over an alphabet Σ is denoted Σ^* and is defined as $\Sigma^* = \{\epsilon\} \cup \{\alpha v \mid \alpha \in \Sigma, v \in \Sigma^*\}$. The length of a string $v \in \Sigma^*$ is denoted $|v|$. The set of sequences over an arbitrary set S is also denoted S^*, the sequence $s_1, \ldots, s_n$ is referred to as an n-tuple. When expedient we may abuse notation and confuse the n-tuple $s_1, \ldots, s_n$ with the string $s_1 \cdots s_n$.

An n by n matrix (all our matrices are square) is an n-tuple of n-tuples $M = ((x_{1,1}, \ldots, x_{1,n}), \ldots, (x_{n,1}, \ldots, x_{n,n}))$ with $x_{i,j} \in \mathbb{N}$ for all $i, j \in [n]$. We say that $x_{i,j}$ is on row i and column j, and denote it by $M_{i,j}$.

A tree t consists of a root node labeled by some symbol $\alpha \in \Sigma$ and a tuple of zero or more direct child subtrees $(t_1, \ldots, t_n)$ (for any $n \in \mathbb{N}$) over the same alphabet. t is denoted by $\alpha[t_1, \ldots, t_n]$. For a tree $\alpha[]$ with zero children we may abbreviate it as simply α. The set of all trees over Σ, denoted by T_Σ, is defined as $T_\Sigma = \Sigma \cup \{\alpha[t_1, \ldots, t_n] \mid \alpha \in \Sigma, n \in \mathbb{N}, t_1, \ldots, t_n \in T_\Sigma\}$.

The set of positions in a tree is defined by a function $pos : T_\Sigma \to 2^{\mathbb{N}^*}$. For any $k \in \mathbb{N}$, including zero, $\alpha \in \Sigma$ and $t_1, \ldots, t_k \in T_\Sigma$ the definition of $pos\big(\alpha[t_1, \ldots, t_k]\big)$ is $\{\epsilon\} \cup \big\{(i, v_1, \ldots, v_n) \mid i \in \{1, \ldots, k\}, (v_1, \ldots, v_n) \in pos(t_i)\big\}$. That is, a position $p \in pos(\alpha[t_1, \ldots, t_n])$ denotes the root note α if $p = \epsilon$, otherwise p is of the form $(i, v_1, \ldots, v_n)$ referring to the position $(v_1, \ldots, v_n)$ in the subtree t_i.

3 The Extended String-to-String Correction Problem

A (pre-existing) problem that we will make use of in the coming proof will now be defined. Later on we will use a reduction from an instance of the *extended string-to-string correction problem* (ESSCP) to our problem to show strong NP-hardness. The ESSCP is known to be NP-complete (problem [SR20] in [4]), shown in the case where the cost of inserts and replacements is made infinite and when swaps and deletes are given a constant cost [10]. The formulation by Wagner in [10] allows arbitrary costs

for deletes and any non-zero cost for swaps, while the formulation in [4] fixes both costs to 1. Here we opt to set the cost of a single swap to 1 and the cost of deletes to 0, this causes no loss of generality, since the number of deletes in a solution is always the difference in length between the source and target strings. The problem definition is divided into three parts, for all $\alpha_1 \cdots \alpha_n \in \Sigma^*$:

Definition 1 (String deletes). *For all* $\{d_1, \ldots, d_m\} \subseteq [n]$ *we define the delete function as* $delete(\alpha_1 \cdots \alpha_n, \{d_1, \ldots, d_m\}) = \alpha_{i_1} \cdots \alpha_{i_{n-m}}$ *where* $i_1 < \ldots < i_{n-m}$ *and* $\{i_1, \ldots, i_{n-m}\} = [n] \setminus \{d_1, \ldots, d_m\}$.

Definition 2 (String swaps). *We define the swap function by letting* $\texttt{swap}(s, \epsilon) = s$ *for all strings* s *and for all* $(s_1, \ldots, s_m) \in [n-1]^*$ *letting*

$$\texttt{swap}(\alpha_1 \cdots \alpha_n, (s_1, \ldots, s_m)) = \texttt{swap}(\alpha_1 \cdots \alpha_{s_1-1}\alpha_{s_1+1}\alpha_{s_1}\alpha_{s_1+2} \cdots \alpha_n, (s_2, \ldots, s_m)).$$

Definition 3 (The delete/swap ESSCP). *An instance of the delete/swap ESSCP (over some alphabet* Σ*) is a tuple* $(S, T, b) \in \Sigma^* \times \Sigma^* \times \mathbb{N}$. *The instance is a "yes" instance (the answer is "yes") if and only if there exists some* $D \subseteq [|S|]$ *and* $W \in [|S| - |D| - 1]^*$ *such that* $\texttt{swap}(\texttt{delete}(S, D), W) = T$ *with* $|W| \leq b$. *We denote the set of all such "yes" instances* ESSCP_{ds}.

There are a couple of important things to notice here.

- The definition is stated so that all deletes happen before any swap. This is not a restriction of the problem, since there is no instance where it is better to delete something after moving it around.
- b is in all interesting instances polynomial in the size of the instance, since all reorderings can be realized in less than n^2 swaps. We therefore, without loss of generality, assume b to be coded in unary in the input, so ESSCP_{ds} is strongly NP-complete.
- Swaps of unrelated symbols can be reordered freely. One recurring example is that if $\texttt{swap}(\alpha_1 \cdots \alpha_n, W)$ is such that the symbol α_i is moved to the end of the string by W we can trivially restructure W to start with the sequence $i, i+1, \ldots, n-1$, without making W longer. That is, if a minimal swap sequence moves the symbol in position i to the last position n then doing this before anything else cannot make the swap sequence longer, since keeping the symbol in the middle of the string for longer serves no purpose.

4 Swap Assignment Problem

Now we will define the first original problem, the swap assignment problem. We will demonstrate that this problem is strongly NP-complete by a reduction from ESSCP_{ds}. This problem will serve as a stepping stone to demonstrate NP-completeness for the tree swap distance problem.

This problem is quite similar to the classical assignment problem [6], except a starting assignment is given, and an optimal assignment is to be reached by swapping adjacent assignments. The swap function is defined exactly as in the string case, when the matrix is viewed as a string of rows.

Definition 4 (Matrix Row Swap). *For an n by n matrix M the swap function is defined by for all $W \in [n-1]^*$ simply viewing the matrix as a string of rows: $(M_{1,1}, \ldots, M_{1,n}) \cdots (M_{n,1}, \ldots, M_{n,n})$ and applying the string swap $\texttt{swap}(M, W)$.*

Definition 5 (The Swap Assignment Problem). *An instance of the swap assignment problem is a tuple (M, b) where $b \in \mathbb{N}$, and M is an n by n matrix. The instance is a "yes" instance if and only if there exists some $W \in [n-1]^*$ such that*

$$b \geq |W| + \sum_{i=1}^{n} \texttt{swap}(M, W)_{i,i}.$$

We denote the set of all such "yes" instances SAP*.*

Let us look at a small instance to better understand the problem.

Example 6. As an example swap assignment problem instance we can take (M, b) with $b = 9$ and M as below.

$$M = \begin{bmatrix} 4 & 5 & 16 & 0 \\ 3 & 4 & 16 & 0 \\ 2 & 3 & 0 & 16 \\ 1 & 2 & 16 & 16 \end{bmatrix} \quad M' = \begin{bmatrix} 4 & 5 & 16 & 0 \\ 1 & 2 & 16 & 16 \\ 2 & 3 & 0 & 16 \\ 3 & 4 & 16 & 0 \end{bmatrix}.$$

Since we can use the swaps $W = 3, 2, 3$ to construct $M' = \texttt{swap}(M, W)$ as shown above, it follows that $(M, b) \in \text{SAP}$. M' has the diagonal sum 6 which together with the three swaps adds up to exactly 9. We could also equivalently solve the problem instance using the swap-sequence $W' = 1, 3, 2, 3$ which produces a diagonal cost of $3 + 2 + 0 + 0 = 5$ but, on the other hand, requires 4 swaps, again giving a total of 9.

The ESSCP_{ds} (Definition 3) can be reduced to the swap assignment problem in a slightly tricky to visualize but functionally straightforward way.

Definition 7 (ESSCP to Swap Assignment Reduction). *Take a delete/swap ESSCP instance $(s_1 \cdots s_n, t_1 \cdots t_m, b)$ (we assume that $m \leq n$, otherwise it is trivial). Then construct a swap assignment problem instance (M, b') where the n by n matrix M is constructed by taking:*

$$M_{i,j} = \begin{cases} 0 & \text{if } j \leq m \text{ and } s_i = t_j, \\ b' + 1 & \text{if } j \leq m \text{ and } s_i \neq t_j. \\ n + i - j & \text{if } j > m, \end{cases},$$

and $b' = b + n(n - m)$.

This definition is not really intuitive, but a short example should explain the idea of how this represents an ESSCP instance.

Example 8. Let us consider the delete/swap ESSCP instance $(aacb, abc, 1)$. This has a fairly simple solution, delete one of the "a" symbols and swap the "b" and "c". The reduction computes $b' = 1 + 4(4 - 3) = 5$ and the matrix

$$M = \begin{bmatrix} 0 & 6 & 6 & 1 \\ 0 & 6 & 6 & 2 \\ 6 & 6 & 0 & 3 \\ 6 & 0 & 6 & 4 \end{bmatrix}.$$

We will look at the left part first, the part that corresponds to the first two cases of the construction. All these cells are set either to 0 or to $b' + 1$, which means that none of the non-zero cells *may ever* be on the diagonal of a solution, since the sum would always be greater than the budget. So, the first three positions on the diagonal (counting from the upper left) must be made zero in a solution, the three corresponds to the length of the target string. The idea is that a zero on the diagonal in this first part corresponds to a correctly matched symbol. The cells on the right-hand side only come into play on the last part of the diagonal, the bottom few rows of the result. The rows moved to the bottom correspond to symbols that get deleted.

The motivation for the weight $n + i - j$ in case 3 of the reduction is that if we wish to delete some symbol in the original string problem we have a fixed cost (zero), but to move a row to the bottom of the matrix has different cost depending on where the row starts out, since different numbers of swaps need to be used. The cost the rows that end up at the bottom contribute to the diagonal is there to counteract this. Let us look at the two ways to solve this instance, see Figure 1. Here we show the

$$\begin{bmatrix} 0&6&6&1\\ 0&6&6&2\\ 6&6&0&3\\ 6&0&6&4 \end{bmatrix} \Rightarrow \begin{bmatrix} 0&6&6&2\\ 0&6&6&1\\ 6&6&0&3\\ 6&0&6&4 \end{bmatrix} \Rightarrow \begin{bmatrix} 0&6&6&2\\ 6&6&0&3\\ 0&6&6&1\\ 6&0&6&4 \end{bmatrix} \Rightarrow \begin{bmatrix} 0&6&6&2\\ 6&6&0&3\\ 6&0&6&4\\ 0&6&6&1 \end{bmatrix} \Rightarrow \begin{bmatrix} 0&6&6&2\\ 6&0&6&4\\ 6&6&0&3\\ 0&6&6&1 \end{bmatrix}$$

Figure 1: A solution for the the swap assignment problem instance produced by reducing from $(aacb, abc, 1) \in \mathrm{ESSCP}_{\mathrm{ds}}$

solution equivalent to deleting the first "a", by swapping the top row down to the bottom with the first three swaps. This row then contributes cost 1 to the diagonal, for a total cost of 4 to get rid of the first symbol. Then we swap the rows that were originally 3 and 4 (going from "acb" to "abc") to move the zeros to the diagonal. The total cost of the solution is 5, which fits the budget b'.

What is key is that the solution can choose to delete any symbol without the cost being different. So let us look at the other possibility, where we delete the second "a" instead, shown in Figure 2. Here we start by swapping the second row, corresponding

$$\begin{bmatrix} 0&6&6&1\\ 0&6&6&2\\ 6&6&0&3\\ 6&0&6&4 \end{bmatrix} \Rightarrow \begin{bmatrix} 0&6&6&1\\ 6&6&0&3\\ 0&6&6&2\\ 6&0&6&4 \end{bmatrix} \Rightarrow \begin{bmatrix} 0&6&6&1\\ 6&6&0&3\\ 6&0&6&4\\ 0&6&6&2 \end{bmatrix} \Rightarrow \begin{bmatrix} 0&6&6&1\\ 6&0&6&4\\ 6&6&0&3\\ 0&6&6&2 \end{bmatrix}$$

Figure 2: An alternative solution for the swap assignment problem instance produced by reducing from $(aacb, abc, 1) \in \mathrm{ESSCP}_{\mathrm{ds}}$

to the second "a" into the last position. This takes only 2 swaps, but this row contributes a cost of 2 to the diagonal, again making the delete cost exactly 4. A final swap of the original row three and four again produces a solution with cost 5.

This illustrates the key property of the construction, deletions are substituted with moving the rows in question into bottom positions, and the costs in the rows are

constructed so that a row that is originally far from the bottom gets a proportionally larger "discount" on the diagonal sum to pay for the extra swaps needed to delete them. The formula for the rightmost column is $n + i - j$, the subtraction of j comes into play when multiple symbols are deleted. Since not all rows can go to the bottom position later deletions will have a shorter distance to travel than the first ones, this is counteracted by the costs being greater in the "discount columns" further left. As a final example see the slightly larger instance in Figure 3.

$$\left[\begin{array}{ccc|cc} 12 & 12 & 0 & 2 & 1 \\ 12 & 0 & 12 & 3 & 2 \\ 0 & 12 & 12 & 4 & 3 \\ 0 & 12 & 12 & 5 & 4 \\ 12 & 12 & 0 & 6 & 5 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|cc} 0 & 12 & 12 & 4 & 3 \\ 12 & 0 & 12 & 3 & 2 \\ 12 & 12 & 0 & 6 & 5 \\ 12 & 12 & 0 & 2 & 1 \\ 0 & 12 & 12 & 5 & 4 \end{array}\right]$$

Figure 3: Reducing $(cbaac, abc, 1) \in \text{ESSCP}_{\text{ds}}$ produces the swap assignment problem instance with the left matrix and budget $b' = 11$. "Deleting" a row ends up with a cost of 5 counting swaps and diagonal cost. On the right is the solution which performs the swaps $4, 1, 2, 3, 1$ for a total cost of 11. This solution corresponds to deleting the last "a", deleting the first "c" and finally swapping the remaining "b" and "a".

Lemma 9. *The reduction in Definition 7 produces a swap assignment problem instance that answers "yes" if and only if the original delete/swap ESSCP instance answers "yes".*

Proof (Sketch). Starting with the "if" direction, take some $(s_1 \cdots s_n, t_1 \cdots t_m, b) \in \text{ESSCP}_{\text{ds}}$. Let the deletes and swaps that solves this instance be $\{d_1, \ldots, d_{n-m}\} \subseteq [n]$ and $W \in [m-1]^*$. Construct (M, b') using the reduction. Assume that $d_1 > d_2 > \cdots > d_{n-m}$ then construct the swaps:

$$W_d = d_1, d_1 + 1, \ldots, n-1, d_2, d_2 + 1, \ldots, n-2, \ldots, d_{n-m}, \ldots, m$$

That is, take row d_1, which corresponds to the last (position-wise) symbol deleted in the original string, and swap it into the last position in the matrix. Then swap row d_2 (second to last deleted position) into the second to last position in the matrix and so on. Now construct $W' = W_d W$ (concatenating the two), after applying the swaps W_d the top m rows in the matrix correspond to the positions which are *not* deleted, and we perform the swaps in W on these.

Now we will just demonstrate that $(M, b') \in \text{SAP}$ using W' as the solution. $|W'| = |W_d| + |W|$ and $|W_d|$ contains $(n-i) - d_i$ swaps to place the row initially at d_i into position $n-i$, for each $i \in [n-m]$. So the row (initially at) d_i will contribute $M_{d_i, n-i}$ to the final diagonal sum. The range of i means that $M_{d_i, n-i} = n + d_i - (n-i) = d_i + i$ (since all these positions are filled by the third case in the construction of M in Definition 7). Taking the swaps and diagonal contribution together each of the d_i rows contribute to the total cost by $(n-i) - d_i + d_i + i = n$, meaning that

$$|W_d| + \sum_{i=m+1}^{n} \texttt{swap}(M, W')_{i,i} = (n-m)n.$$

This establishes that $b' = b + (n-m)n \geq |W'| + \sum_{i=m+1}^{n} \texttt{swap}(M, W')_{i,i} = |W| + (n-m)n$, since $b \geq |W|$ and $|W'| = |W_d| + |W|$.

All that needs to be added is the remainder of the diagonal, so next we show that $\sum_{i=1}^{m} \texttt{swap}(M, W')_{i,i}$ is zero. Take $M' = \texttt{swap}(M, W_d)$ and $S' = \texttt{delete}(s_1 \cdots s_n, D)$ and simply note that if the symbol in position i in S' started out in position l then row i in M' started out in position l in M. The next step for both S' and M' is to apply W, meaning that row $j \in [m]$ in the matrix started out as row i if and only if symbol in position j in the final string was originally s_i. Since this is a solution for the ESSCP instance this means that $s_i = t_j$ which means that row i in M ends up in position j in $\texttt{swap}(M, W')$ if and only if $s_i = t_j$. It follows that the new row contributes $M_{i,j}$ to the diagonal, and the construction of M is such that set $M_{i,j} = 0$ when $s_i = t_j$.

Since we showed that $b' \geq |W'| + \sum_{i=m+1}^{n} \texttt{swap}(M, W')_{i,i}$ above and showed that $\sum_{i=1}^{m} \texttt{swap}(M, W')_{i,i} = 0$ here it follows that $b' \geq |W'| + \sum_{i=1}^{n} \texttt{swap}(M, W')_{i,i}$ so $(M, b') \in \text{SAP}$.

The "only if" direction remains but works in a very similar way. Assume that $(M, b') \in \text{SAP}$ is constructed from some delete/swap ESSCP instance (S, T, b). Let W' be the swaps that solve (M, b'). Notice that if such a solution W' exists then a solution exists which has the structure $W' = W_d W$ (that is, which first swaps all the $n - m$ bottom rows into position), if row i is going to be swapped into position n nothing can be gained by not doing so as the first thing in the swap sequence. Using this we can extract the solution to the string problem instance, deleting the symbols corresponding to rows swapped below the mth row. The solution to (M, b') also cannot do better than the fixed cost $(n-m)(n-1)$ for swaps and diagonal of these bottom rows, and it has to place the top m rows so that they all contribute zero to the diagonal (all other positions being $b' + 1$ which is impossible in a solution), which corresponds directly to matching symbols correctly. □

Corollary 10. *The swap assignment problem is strongly NP-complete.*

This follows since ESSCP_{ds} is strongly NP-complete and the reduction constructs a polynomially sized matrix containing numbers that are all bounded by a polynomial in the original instance (recall that b is polynomial in all relevant cases and assumed to be unary). The problem is in NP since no swap sequence ever needs to be longer than n^2, allowing W' to be guessed.

5 Swap Even-Cost Assignment Problem

Now we will define a very minor restriction on the swap assignment problem. This will turn out to be key to make the final reduction to the tree swap distance problem simple.

Definition 11. *Let $2 \,|\, x$ denote that x is even ($x \in \{0, 2, 4, 6, \ldots\}$), let $2 \nmid x$ denote that x is odd.*

Definition 12 (Swap Even-Cost Assignment Problem). *An instance of the swap even-cost assignment problem is a swap assignment problem instance (M, b) such that $2 \,|\, M_{i,j}$ for all $i, j \in [n]$. The answer to (M, b) is "yes" if and only if $(M, b) \in \text{SAP}$. We denote the set of all "yes" instances as* SecAP*.*

We will quickly establish that all swap assignment problem instances have an equivalent swap even-cost assignment problem instance.

Definition 13. *Let* $h(x) = \left\lceil \frac{x}{2} \right\rceil$.

Definition 14 (Reducing SAP to SecAP). *Let* (M, b) *be an instance of the swap assignment problem with* M *an* n *by* n *matrix, we then construct* (M', b')*, where* M' *is a* $2n$ *by* $2n$ *matrix, by letting* $b' = b + \frac{n(n-1)}{2}$ *and taking*

$$M'_{i,j} = \begin{cases} M_{i,h(j)} & \text{if } i \leq n,\ 2 \nmid j \text{ and } 2 | M_{i,h(j)}, \\ b'' & \text{if } i \leq n,\ 2 \nmid j \text{ and } 2 \nmid M_{i,h(j)}, \\ M_{i,h(j)} - 1 & \text{if } i \leq n,\ 2 | j \text{ and } 2 \nmid M_{i,h(j)}, \\ b'' & \text{if } i \leq n,\ 2 | j \text{ and } 2 | M_{i,h(j)}, \\ 0 & \text{if } i > n \text{ and } h(j) = i - n, \\ b'' & \text{if } i > n \text{ and } h(j) \neq i - n, \end{cases}$$

where b'' *is the smallest even number strictly larger than* b'.

This definition is also a bit daunting but the underlying thinking is fairly straightforward, let us look at an example.

Example 15. We will start with an instance of the swap assignment problem instance (M, b), where $b = 11$ and M is shown on the left in Figure 4. For this example $b' = 14$,

$$M = \begin{bmatrix} 2 & 3 & 3 \\ 9 & 4 & 12 \\ 1 & 2 & 8 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 16 & 16 & 2 & 16 & 2 \\ 16 & 8 & 4 & 16 & 12 & 16 \\ 16 & 0 & 2 & 16 & 8 & 16 \\ 0 & 0 & 16 & 16 & 16 & 16 \\ 16 & 16 & 0 & 0 & 16 & 16 \\ 16 & 16 & 16 & 16 & 0 & 0 \end{bmatrix}$$

Figure 4: Example of applying the even-cost reduction to a swap assignment problem instance

so $b'' = 16$. Let us look at the upper half of the matrix first. The thing to notice about this part is that for all $i, j \in [n]$ there are for each pair $(M_{2i-1,j}, M_{2i,j})$ only two cases, either the pair is $(M_{i,j}, 16)$ if $M_{i,j}$ was even, or it is $(16, M_{i,j} - 1)$ if $M_{i,j}$ was odd.

This starts making sense when we look at the lower half of the matrix, which is filled with rows such that for each $j \in [n]$ the row at position $n + j$ can only be in either position $2j - 1$ or $2j$ in a valid solution (since that brings the rows zero positions to the diagonal, and b'' is guaranteed to be more than the budget). This means that any valid solution will be structured so that for each $j \in [n]$ one of the positions $2j - 1$ and $2j$ contains the row originally in position $n + j$ (in all other positions it would contribute b'' to the diagonal making the solution impossible) and the other position contains some row originally in the top half (since all rows from the bottom half are already accounted for). The $\frac{n(n-1)}{2}$ part of the budget is exactly enough to pay for the minimal such interspersing (where the row from the top half is the one at the $2j - 1$ position since that is closer).

Let $i \in [n]$ be the initial position of the row from the top that ends up in position $2j-1$ or $2j$, this row is supposed to simulate the cost $M_{i,j}$ on the diagonal. If $M_{i,j}$ is even this is easy, the row can be placed at position $2j-1$ (since it will have $M'_{i,2j-1} = M_{i,j}$), if $M_{i,j}$ contained an odd number however the construction has made $M_{i,2j-1} = b''$, which forces the solution to take an extra swap to bring the row to position $2j$. This extra swap fixes the cost that was lost when the construction rounded down $M'_{i,2j} = M_{i,j} - 1$.

To make this more visual see Figure 5. Since this solution involves a total of

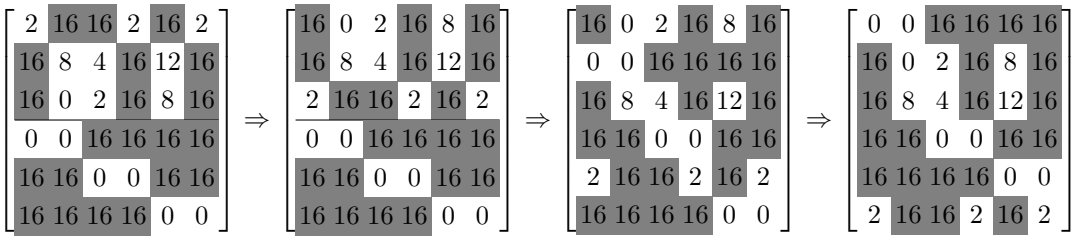

Figure 5: Some steps of the solution of the problem instance in Figure 4

seven swaps several are done in each step. Let us first note that a solution for the original (pre-reduction) instance in Figure 4 is to swap $2, 1, 2$, giving a diagonal sum of $1 + 4 + 3 = 8$ and a total solution cost of 11. In Figure 5 we have the original reduced matrix on the left, in the first step we do the same three swaps $2, 1, 2$. In the next step we intersperse the rows from the bottom half with the top with the swaps $3, 2, 4$. This however leaves us with 16 in two places on the diagonal, and have to finish with the swaps $1, 4$. These last swaps are key. Notice how the diagonal in the original instance ended up being $1 + 4 + 3$, the first and last positions are odd. The construction took these odd numbers, rounded them down to something even and placed this rounded result on the right side of its horizontal "pair" in the top row. This forces the solution to do extra swaps to bring the rows down one step further, paying the cost that was removed by the rounding. In total the solution here makes 8 swaps, and has a diagonal sum of 6, for a total cost of 14, exactly the budget b'.

Lemma 16. *For every swap assignment problem instance (M, b) (M is n by n) the reduction in Definition 14 produces a swap even-cost assignment problem instance (M', b') such that $(M', b') \in$ SecAP if and only if $(M, b) \in$ SAP.*

Proof (Sketch). Assume that $(M, b) \in$ SAP. Let W be a swap sequence that solves (M, b). Then construct a (minimal) swap sequence W_i such that

$$\texttt{swap}(a_1 \cdots a_n b_1 \cdots b_n, W_i) = a_1 b_1 a_2 b_2 \cdots a_n b_n,$$

and, let $W_o = o_1 \cdots, o_m$ be such that $o_1 < \cdots < o_m$ and $2 \nmid \texttt{swap}(M, W)_{i,i}$ if and only if $i \in \{o_1, \ldots, o_m\}$. Then $W' = W W_i W_o$ (the concatenation) is a solution for (M', b'). This sequence of swaps being a solution is quickly established, noting that $|W_i| = \frac{n(n-1)}{2}$ which accounts for the difference between b' and b, and then noting that the construction makes all the swaps in W_o necessary.

The other direction amounts to assuming the existence of W' and then extracting the W part which concerns the internal order of the n first rows. □

Corollary 17. *The swap even-cost assignment problem is strongly NP-complete.*

This follows from the above. The reduction from the strongly NP-complete swap assignment problem is clearly polynomial, the matrix dimensions are doubled and the values in the matrix grow on the order of $\mathcal{O}(n^2)$. The problem is in NP, since SecAP is simply SAP with inputs restricted to even numbers.

6 Tree Swap Distance Problem

This section will reach the goal of the paper, defining the tree swap distance problem and then demonstrating that it is strongly NP-complete by a reduction from SecAP. Let us define the problem.

Definition 18 (Tree Swap). *Take any tree $t = \alpha[t_1, \ldots, t_n] \in T_\Sigma$ and any $P = (p_1, \ldots, p_m) \in pos(t)$ such that $(p_1, \ldots, p_{m-1}, (p_m + 1)) \in pos(t)$. Then define the single-swap function*

$$\mathtt{swap}_1(t, P) = \begin{cases} \alpha[t_1, \ldots, t_{p_1-1}, \mathtt{swap}_1(t_{p_1}, (p_2, \ldots, p_m)), t_{p_1+1}, \ldots, t_n] & \text{if } m > 1, \\ \alpha[t_1, \ldots, t_{p_1-1}, t_{p_1+1}, t_{p_1}, t_{p_1+2}, \ldots, t_n] & \text{otherwise.} \end{cases}$$

The full swap function is for (appropriate) positions $P_1, \ldots, P_p$ defined as

$$\mathtt{swap}(t, (P_1, \ldots, P_p)) = \mathtt{swap}_1(\ldots \mathtt{swap}_1(\mathtt{swap}_1(t, P_1), P_2) \ldots, P_p).$$

The definition of swaps for trees is slightly unwieldy, but the swap function takes a tree and a sequence of tree positions (which are integer sequences). The positions identify, in order, the subtree which should next swap position with its sibling immediately to the right. Notice that P_i for $i > 1$ does not refer to a position in the tree t but to a position in an intermediary tree, it may be that $P_i \notin pos(t)$. An example is shown in Figure 6.

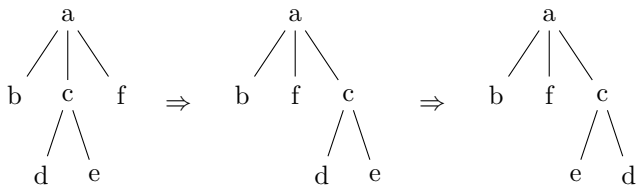

Figure 6: An example of applying the tree swaps $((2), (3, 1))$ to a small tree. That is, going from the first to second tree we swap the position 2, referring to the second child of the root, next the position $(3, 1)$ is swapped, referring to the first child of the rightmost child subtree of the root.

The definition of the tree swap distance problem now follows a familiar formula.

Definition 19 (The Tree Swap Distance Problem). *An instance of the tree swap distance problem is a tuple (t, t', b) where $t \in T_\Sigma$ is the start tree, $t' \in T_\Sigma$ is the target tree and $b \in \mathbb{N}$ is the budget. The instance is a "yes" instance if and only if there exists some $P_1 \in \mathbb{N}^*, \ldots, P_n \in \mathbb{N}^*$ such that $n \leq b$ and $t' = \mathtt{swap}(t, (P_1, \ldots, P_n))$. We denote the set of all such "yes" instances* TSwD.

The next definition is used to make it easier to talk about minimal swap sequences.

Definition 20 (Minimal budget for TSwD). *For all $t, t' \in T_\Sigma$ let $mincost(t, t') = b$, where $b \in \mathbb{N}$ is the smallest number for which $(t, t', b) \in$ TSwD. If no such number exists let $b = \infty$.*

The reduction from SecAP to TSwD requires some building blocks. A visual example of the different types of notation defined below is shown later in Figure 8.

Definition 21 (Number Tree). *Assume that $0, 1 \in \Sigma$. For some symbol $\alpha \in \Sigma$ and $x, y \in \mathbb{N}$ such that $x \leq y$ we let $\alpha[x : y]$ denote the tree $\alpha[p_1, \ldots, p_{y+1}]$ where $p_i = 0$ for all $i \neq x + 1$ and $p_{x+1} = 1$.*

For example, $\alpha[2 : 3] = \alpha[0, 0, 1, 0]$. We call these trees "number trees". Notice that for all $x, x', y \in \mathbb{N}$ such that $x \leq y$ and $x' \leq y$ it holds that $mincost(\alpha[x : y], \alpha[x' : y]) = |x - x'|$. That is, the minimum number of swaps needed to turn $\alpha[x : y]$ into $\alpha[x' : y]$ is exactly $|x - x'|$. The tree $\alpha[x : y]$ serves the purpose to represent the number x, with the minimal swap distance to any other $\alpha[x' : y]$ being the absolute difference between x and x'.

Definition 22 (Number Trees with Neutral Elements). *Assume that for each $\alpha \in \Sigma$ there exists a distinct $\alpha' \in \Sigma$. Then for all $x, y \in \{0, 2, 4, 6, \ldots\}$ let $\alpha\langle x : y\rangle$ denote the following special tree.*

$$\alpha\langle x : y\rangle = \alpha\left[\alpha\left[\frac{x}{2} : \frac{y}{2}\right], \alpha'\left[\frac{y-x}{2} : \frac{y}{2}\right]\right].$$

Additionally let $\alpha\langle \bot : y\rangle$ denote the special tree $\alpha\left[\alpha\left[0 : \frac{y}{2}\right], \alpha'\left[0 : \frac{y}{2}\right]\right]$, called a "neutral" tree.

So, for example $\alpha\langle 2 : 6\rangle$ is the tree $\alpha[\alpha[0, 1, 0, 0], \alpha'[0, 0, 1, 0]]$. These trees have the property that for all $x, x', y \in \{0, 2, 4, 6, \ldots\}$ it holds that $mincost(\alpha\langle x : y\rangle, \alpha\langle x' : y\rangle) = |x - x'|$. This should not be a surprise, these trees behave like the earlier number trees, only the necessary swaps are split across two subtrees, and we lose the capability to represent odd numbers in the process. The gain lies in the neutral trees, it holds that $mincost(\alpha\langle \bot : y\rangle, \alpha\langle x : y\rangle) = \frac{y}{2}$ completely *independently of the value x*.

Definition 23 (Multi-number Trees). *For some $\alpha \in \Sigma$ and $k \in \mathbb{N}$ assume that we have the distinct symbols $\alpha_1, \ldots, \alpha_k \in \Sigma$. Then, for all $x_1, \ldots, x_k \in \mathbb{N} \cup \{\bot\}$, such that either $x_i \leq y$ or $x_i = \bot$ for all $i \in [n]$, let $\alpha\langle (x_1, \ldots x_k) : y\rangle$ denote the tree*

$$\alpha[\alpha_1\langle x_1 : y\rangle, \ldots, \alpha_k\langle x_k : y\rangle].$$

This means that

$$mincost(\alpha\langle (x_1, \ldots, x_n) : y\rangle, \alpha\langle (x'_1, \ldots, x'_n) : y\rangle) = \sum_{i=1}^{n} |x_i - x'_i|,$$

for all $x_1, x'_1, \ldots, x_n, x'_n, y \in \mathbb{N}$ such that $x_i \leq y$ and $x'_i \leq y$ for all $i \in [n]$.

Now all the building blocks necessary to reduce a swap even-cost assignment problem instance to a tree swap problem instance are ready.

Definition 24 (Reducing SecAP to TSwD). *Let (M, b) be an instance of the swap even-cost assignment problem as in Definition 12. We then construct the instance (t, t', b') of the tree swap distance problem as follows. Assume that M is an n by n matrix, let τ be the largest integer that occurs in M. Then let $b' = b + \frac{n(n-1)\tau}{2}$ and construct*

$$\begin{aligned} t = \alpha[\, &\beta\langle(M_{1,1}, \ldots, M_{1,n}) : \tau\rangle, \\ &\beta\langle(M_{2,1}, \ldots, M_{2,n}) : \tau\rangle, \\ &\vdots \\ &\beta\langle(M_{n,1}, \ldots, M_{n,n}) : \tau\rangle], \end{aligned}$$

and

$$\begin{aligned} t' = \alpha[&\beta\langle(0, \bot, \bot, \ldots, \bot) : \tau\rangle, \\ &\beta\langle(\bot, 0, \bot, \ldots, \bot) : \tau\rangle, \\ &\vdots \\ &\beta\langle(\bot, \bot, \ldots, \bot, 0) : \tau\rangle], \end{aligned}$$

that is, $t' = \alpha[t_1, \ldots, t_n]$ such that for all $i \in [n]$ we have $t_i = \beta\langle(x_1, \ldots, x_n) : \tau\rangle$ where $x_j = \bot$ for all $j \neq i$ and $x_i = 0$.

The dense notation may make this reduction hard to visualize, let us look at an example.

Example 25. Let (M, b) be an instance of the swap even-cost assignment problem, letting $b = 3$ and

$$M = \begin{bmatrix} 4 & 0 \\ 2 & 2 \end{bmatrix}.$$

Now we construct the tree swap distance problem instance (t, t', b') by applying the reduction from Definition 24. From M we see that $\tau = 4$, so the budget becomes $b' = 3 + \frac{2(2-1)4}{2} = 7$. The constructed trees are

$$\begin{aligned} t &= \alpha[\beta\langle(4, 0) : 4\rangle, \beta\langle(2, 2) : 4\rangle], \\ t' &= \alpha[\beta\langle(0, \bot) : 4\rangle, \beta\langle(\bot, 0) : 4\rangle]. \end{aligned}$$

To get past the notation the full tree t is shown in Figure 7, and the tree t' (as well as a breakdown of which subtrees correspond to which piece of notation) is shown in Figure 8.

Using these figures it is not hard to see how the solutions to (M, b) and (t, t', b') correspond to each other. (M, b) has a single solution, swapping the two rows (which gives a diagonal sum of 2, for a total cost of 3, which is exactly the budget), making no swap is not an option since the initial diagonal sum is 6, which is over the budget.

The decision to swap the rows in M or not corresponds to the decision whether or not to swap the $\beta\langle\ldots\rangle$-subtrees in t. The reader can easily verify by inspecting Figure 7 and 8 that it takes 10 swaps to move the 0/1 nodes around to match t' if we do not swap the $\beta\langle\ldots\rangle$-subtrees first, which is over the budget (in fact, it is over the budget by the same amount as the initial order of M is for that instance). If the two $\beta\langle\ldots\rangle$-subtrees are swapped however, we can reorder the 0/1 nodes in the resulting tree in only 6 swaps, for a total cost of 7, exactly the budget b'.

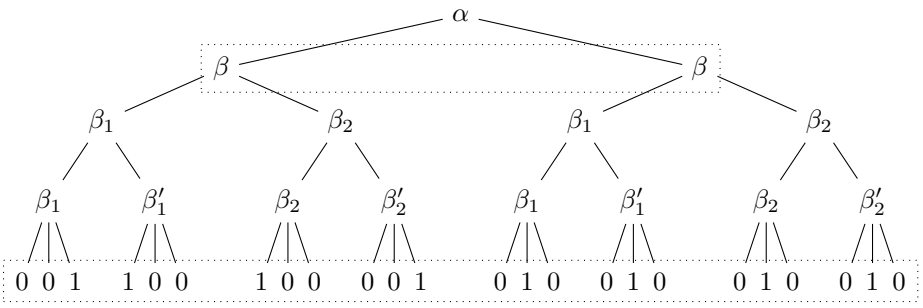

Figure 7: The tree t constructed in the reduction in Example 25. Notice that any solution only needs to perform swaps on the nodes in the dotted rectangles, all other nodes are already in their only possible internal order (compare to t' in Figure 8).

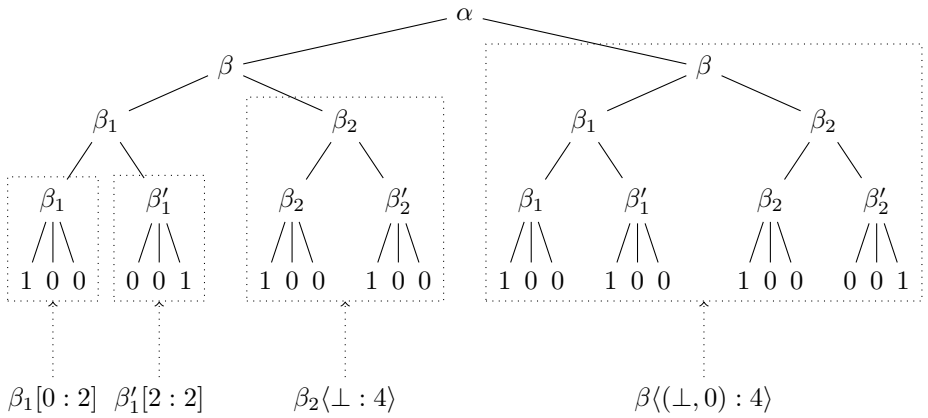

Figure 8: The tree t' constructed in the reduction in Example 25. The dotted arrows shows the notation we use to describe the indicated parts of the tree.

Hopefully the example has clarified the general idea of this reduction, but a proof sketch follows which further illustrates how it functions in the general case.

Lemma 26. *For every swap even-cost assignment problem instance (M, b) and tree swap distance problem instance (t, t', b') constructed from (M, b) by the reduction in Definition 24 it holds that $(t, t', b) \in$ TSwD if and only if $(M, b) \in$ SecAP.*

Proof (Sketch). We reuse the notation of the reduction. First notice that there are only two levels of swapping to consider in t. The immediate subtrees can be reordered since all are of the β multi-number kind, this is the interesting part. In addition the leaves will be swapped to move around the 0/1 sequences that are there to represent numbers, but this is abstracted by our number trees and can only be done in one trivial way once the top-level swaps are decided. The nodes in between are marked with distinct symbols.

Now let us look at the sub-subtrees in t'. There are n^2 of them, organized into n subtrees, each of which represents a row. For each $i \in [n]$ look at position i, i in t', this tree is of the form $\beta_i\langle 0 : \tau\rangle$, whereas for all $i, j \in [n]$ such that $i \neq j$ the subtree at position i, j is of the form $\beta_j\langle \bot : \tau\rangle$. These $n(n-1)$ trees will be matched up with some β_j sub-subtree in t at a constant cost of $\frac{\tau}{2}$ each, incurring a constant and unavoidable cost of $\frac{n(n-1)\tau}{2}$, leaving exactly b of the budget for the remainder.

This leaves the n "diagonal" subtrees of the form $\beta_i\langle 0 : \tau\rangle$ in t'. Assume that W in M moves row i into position j, incurring some swap cost and a diagonal cost of $M_{i,j}$. If we apply W directly to t this would move subtree $\beta\langle M_{i,1}, \ldots, M_{i,n}\rangle$ into position to match the tree in t' that contains the zero number tree $\beta_j\langle 0 : \tau\rangle$ in position j. This means that the cost incurred, beyond the already accounted for constant cost associated with the $n-1$ neutral trees will be $\mathit{mincost}(\beta_j\langle M_{i,j} : \tau\rangle, \beta_j\langle 0 : \tau\rangle)$, which is exactly $M_{i,j}$ by the construction of the number trees. So, to recap, applying W at the top level leaves us with the constant cost of $\frac{n(n-1)\tau}{2}$ plus $|W|$ plus $M_{i,j}$ for each row moved from position i to position j by W. Which is exactly the same cost that applying W in M incurs plus $\frac{n(n-1)\tau}{2}$, and since $b' = b + \frac{n(n-1)\tau}{2}$ this makes the problem instances equivalent. We did the argument starting from W, but we can trivially extract the swaps which deal with the immediate subtrees in t from a solution to (t, t', b'), making the other direction very straightforward. □

Corollary 27. *The tree swap distance problem is strongly NP-complete.*

As before the problem being in NP is trivial since the swap sequence never needs to be longer than n^2 so we may guess it. The reduction being polynomial is not hard to see, though the details become somewhat lengthy. There are on the order of $\mathcal{O}(\tau n^2)$ nodes in the trees, but SecAP is strongly NP-complete so this unary representation is not problematic.

7 Conclusion

Treating a problem where the only conclusion is negative, the problem being intractable, is never quite the ideal outcome. On the other hand it was already known that tree edit distance with subtree movement is problematic, and the efforts to integrate limited forms of swaps have been ongoing for some time. As such it is useful to establish that swaps are *inherently* problematic in trees. This hints that better results may be achieved if one considers simpler measures, such as linear distance, where all subtrees are reordered simultaneously and the cost of moving a subtree from position i to position j is exactly $|i-j|$ independent of whether the trees in between are moved. This would allow the Hungarian algorithm [6] to be leveraged in the tree case, giving a polynomial algorithm.

The problem itself may also be useful for complexity analysis of other swap problems, since it is at its core very simple both to explain and intuitively understand.

Hopefully this rather fundamental problem being proven NP-complete will also serve as a useful stepping stone for other complexity-theoretical work.

References

1. D. T. Barnard, G. Clarke, and N. Duncan: *Tree-to-tree correction for document trees*, Tech. Rep. 1995-372, Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, 1995.
2. P. Bille: *A survey on tree edit distance and related problems.* Theor. Comput. Sci., 337(1-3) 2005, pp. 217–239.
3. F. J. Damerau: *A technique for computer detection and correction of spelling errors.* Commun. ACM, 7(3) 1964, pp. 171–176.

4. M. R. Garey and D. S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

5. P. N. Klein: *Computing the edit-distance between unrooted ordered trees*, in In Proceedings of the 6th annual European Symposium on Algorithms (ESA, Springer-Verlag, 1998, pp. 91–102.

6. H. W. Kuhn: *The hungarian method for the assignment problem.* Naval Research Logistics Quarterly, 2(1-2) 1955, pp. 83–97.

7. V. I. Levenshtein: *Binary codes capable of correcting deletions, insertions, and reversals.* Soviet Physics Doklady, 10(8) 1966, pp. 707–710.

8. S. M. Selkow: *The tree-to-tree editing problem.* Inf. Process. Lett., 6(6) 1977, pp. 184–186.

9. K.-C. Tai: *The tree-to-tree correction problem.* J. ACM, 26 July 1979, pp. 422–433.

10. R. A. Wagner: *On the complexity of the extended string-to-string correction problem*, in STOC '75: Proceedings of seventh annual ACM symposium on Theory of computing, New York, NY, USA, 1975, ACM, pp. 218–223.

11. R. A. Wagner and R. Lowrance: *An extension of the string-to-string correction problem.* J. ACM, 22(2) 1975, pp. 177–183.

12. K. Zhang and D. Shasha: *Simple fast algorithms for the editing distance between trees and related problems.* SIAM J. Comput., 18(6) 1989, pp. 1245–1262.

13. K. Zhang, R. Statman, and D. Shasha: *On the editing distance between unordered labeled trees.* Information Processing Letters, 42(3) 1992, pp. 133 – 139.