ISSN 1064-5624, Doklady Mathematics, 2024, Vol. 110, No. 2, pp. 373–378. © Pleiades Publishing, Ltd., 2024.
ISSN 1064-5624, Doklady Mathematics, 2024. © Pleiades Publishing, Ltd., 2024.

An Exact Quadratic Algorithm for the Shortest Tree Transformation

K. Yu. Gorbunov[a],* **and V. A. Lyubetsky**[a,b],**

Presented by Academician of the RAS A.L. Semenov

Received January 24, 2024; revised August 8, 2024; accepted August 8, 2024

Abstract—The article proposes a new exact algorithm of quadratic complexity that solves the problem of the shortest transformation ("alignment") of one weighted tree into another, taking into account arbitrary costs of operations on trees. Three operations are considered: adding vertex deletions to an edge or root of a tree and shifting a subtree with deletions.

Keywords: discrete optimization, shortest tree transformation, operations on a tree, operation cost, exact algorithm, quadratic complexity algorithm, tree alignment

DOI: 10.1134/S1064562424702259

1. INTRODUCTION AND FORMULATION OF THE PROBLEM

The Hamming distance between two words of identical length in a fixed finite alphabet is widely used. Often, words of not necessarily equal length, as well as operations, specific to a certain application are considered. Operations sequentially transform one given word into another; the set of operations is preliminarily chosen and fixed depending on the application under consideration. The distance between two words is defined as the length of the shortest sequence of admissible operations transforming one word into another. Moreover, every operation is usually assigned a strictly positive rational number, which is called the operation cost. Accordingly, the distance is defined as the *minimum* of the sum of the costs of the operations that sequentially transform one given word into another. This distance is not necessarily symmetric and is called the edit distance, or Levenshtein distance. The shortest transformation problem consists of constructing an efficient algorithm for finding this minimum and, most importantly, a sequence of operations on which the minimum is reached. This problem can be solved by numerous dynamic programming algorithms with quadratic running time [1], Chapter 11. A chain of operations that minimizes the sum of operation costs is called the *shortest*. The efficiency of an algorithm is understood as the proof of its exactness (or a nontrivial upper bound for its error) together with the proof of a low-degree polynomial bound for its running time. In applications, the shortest transformation problem often arises for finite graphs defined by a fixed property, rather than by words. Examples are the problem for weighted directed chain-cycle graphs considered in [2] and the problem for weighted rooted trees considered in this paper. Of course, the set of graph-transforming operations depends on the application and graphs under consideration.

In this paper, such an algorithm is constructed for rooted trees all of whose vertices are labeled by letters or a minus sign; these labels are called the *type* or *deletion*, respectively, of the vertex to which they are uniquely assigned. We are given weighted trees and a matrix consisting of arbitrary rational numbers expressing the *similarity* of two types (type–type correspondence) and *penalties* for type–deletion (and vice versa) and deletion–deletion correspondences. The similarity of types can be expressed by any number, but penalties are usually specified by nonpositive numbers. The penalty can depend on the type and position of a vertex on a tree. In bioinformatics applications, such trees are often called cell lineage trees. In [3] for the first two of the three operations considered below, a computer program, called mDELTA [4], was proposed, which solves the shortest transformation problem. The lack of the third operation is essential: for example, without it, a leaf is necessarily transformed into a leaf, which is not always the case in applications. In this paper, allowing all three opera-

[a] *Institute for Information Transmission Problems of the Russian Academy of Sciences (Kharkevich Institute), Moscow, Russia*
[b] *Lomonosov Moscow State University, Moscow, Russia*
**e-mail: gorbunov@iitp.ru*
***e-mail: lyubetsk@iitp.ru*

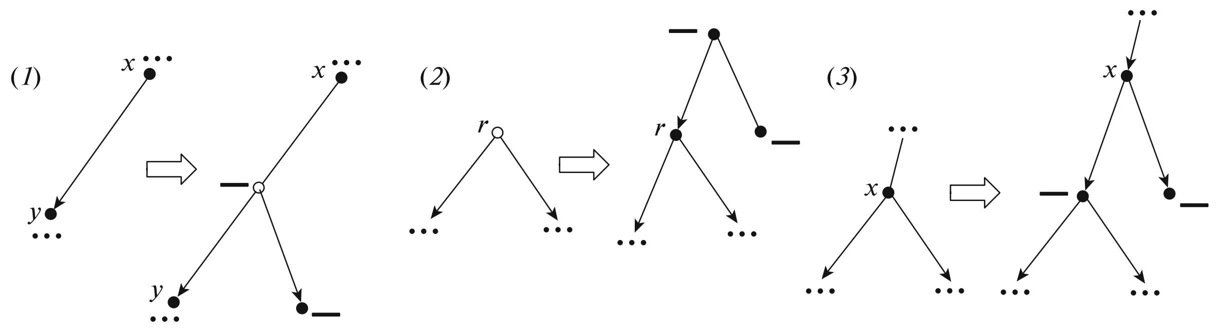

Fig. 1. (*1*) On the edge (x, y) a vertex deletion is added together with its leaf deletion. (*2*) Above the root r, a new root deletion is added together with its leaf deletion. Nothing changes below r. (*3*) At a vertex (including the root or a leaf) labeled by x, the label x is replaced by a deletion and incidently above it a new vertex with label x and a leaf deletion incident to it are added. Nothing changes above the new position of x and below the corresponding new deletion.

tions in the shortest transformation problem, we describe an exact algorithm of quadratic complexity that differs significantly from the mDELTA algorithm. By analogy with algorithms for words, in the tree transformation problem, we say that an algorithm constructs an *alignment* of two original trees for given operation costs.

2. DEFINITION OF OPERATIONS ON ROOTED TREES

The following three operations on trees are allowed; here, x and y denote any type or deletion (see Fig. 1). Any chain G of operations isomorphically embeds each tree into the subsequent one.

(1) On an edge (x, y), add a vertex deletion, together with its leaf deletion, Fig. 1 (operation *1*). It should be emphasized that all operations expand the original tree T by some number of *new deletions* assigned to *new* vertices in T located above its root r, below it, or incomparably with r (it is assumed that trees grow downward from the root).

(2) Above the root r, add a new root deletion, together with its leaf deletion, Fig. 1 (operation *2*). Nothing changes below r.

(3) At a vertex (including the root or a leaf) labeled by x, replace x by a deletion and above incidently add a new vertex with label x and a leaf deletion incident to it. Nothing changes above the new label x and below the corresponding new deletion, Fig. 1 (operation *3*).

Thus, the tree transformation problem can be reformulated as the following *alignment problem*. Given two binary trees T_1 and T_2, use three operations defined above to transform T_1 and T_2 into binary trees T_1' and T_2', $T_1 \subseteq T_1'$, $T_2 \subseteq T_2'$, that are topologically (i.e., without their labels) isomorphic to each other so as to *maximize* the quality of the pair $\{T_1', T_2'\}$ or, in other words, the quality of the isomorphism f: $T_1' \to T_2'$, which is denoted by $H(T_1, T_2) = H(f)$. The *quality* of $\{T_1', T_2'\}$ is defined as the sum (with respect to the isomorphism of all vertices in T_1' and T_2') of type–type similarities plus penalties for type–deletion (or vice versa) and deletion–deletion. In our algorithm, quality is computed by induction. These two chains of operations, $T_1 \to T_1'$ and $T_2 \to T_2'$, extending the original trees T_1 and T_2 to T_1' and T_2', together with the isomorphism f: $T_1' \to T_2'$, are called an *alignment* of T_1 and T_2. The solution of the alignment problem trivially implies the solution of the transformation problem: we transform T_1 into T_1', pass to T_2' by applying f, and we delete all new deletions and vertices added by the algorithm, joining the corresponding edges.

3. ALIGNMENT ALGORITHM

Let us describe the algorithm assuming that deletion–deletion has zero cost in the similarity matrix, while type–deletion (and vice versa) have the same cost independent of the type and arrangement on trees. No generality of the algorithm or the proof is lost under this assumption. The result of replacing the root type by a deletion in a tree T is called a *simplification* of T and is denoted by T^-.

Thus, given two rooted binary trees T_1 and T_2, consider the set D_1 of trees consisting of all subtrees in T_1 and their simplifications. The set D_2 is defined in a similar manner. Let $R, S \in D_i$, $i = 1$ or 2; *R is contained* in S ($R \subseteq S$) if R is a subtree in S or a simplification of a subtree in S. For uniformity, it is convenient to assume that each leaf of each tree in D_i is supplemented with two empty child subtrees with empty edges to a leaf (such empty extensions of leaves are not shown in the figures). A *tree of deletions* is any tree in which all labels are deletions, except for empty subtrees. We *define* a set P of (unordered) pairs consisting of trees $R_1 \in D_1$ and $R_2 \in D_2$, together with the following *partial order* on pairs: $\{R_1, R_2\} \le \{S_1, S_2\}$ if R_1 is con-

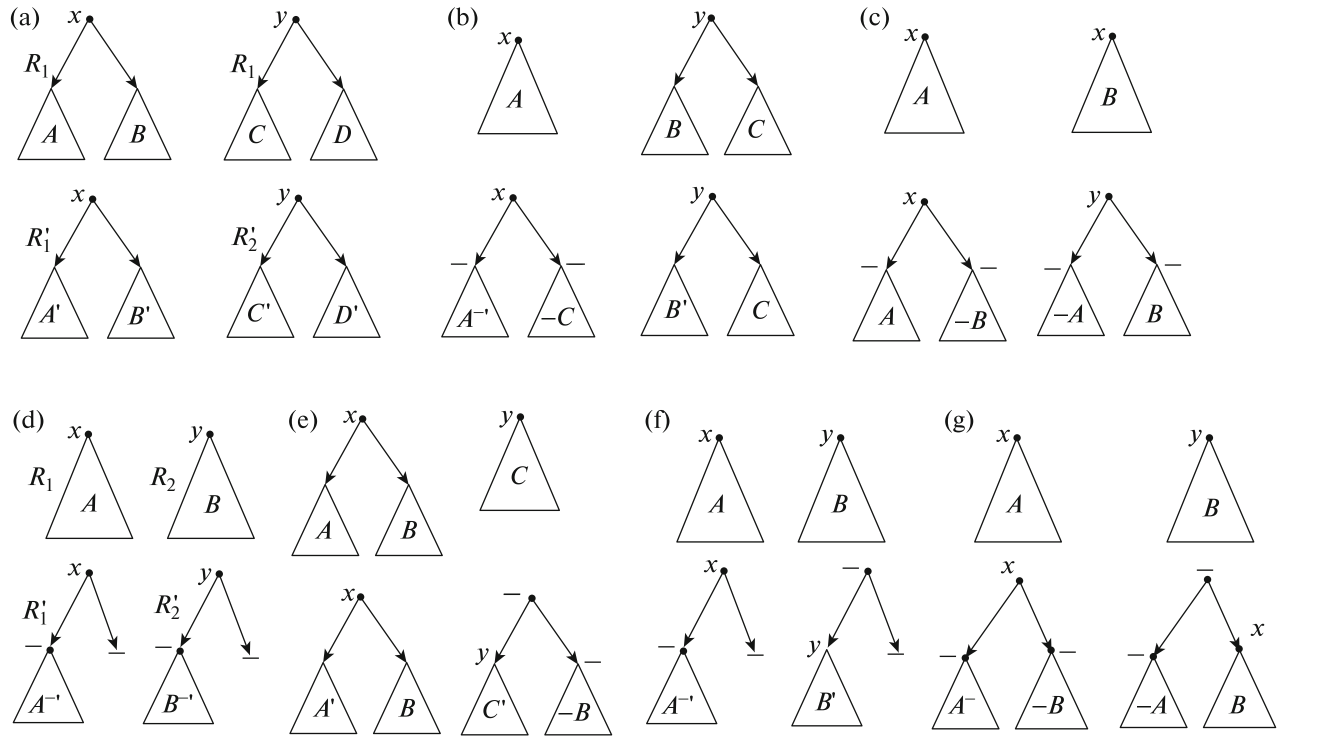

Fig. 2. Transformations $R_1 \to R_1'$ and $R_2 \to R_2'$ of the pair $\{R_1, R_2\}$, computations of the qualities $H(R_1, R_2)$, and links to the transformation at which the maximum of these qualities is reached.

tained in $S_1 \in D_1$ and R_2 is contained in $S_2 \in D_2$. Fix any *linear order* extending the partial order on pairs $\{R_1, R_2\}$. The algorithm performs forward and backward induction on this linear order.

Base of induction. Let $k(R)$ denote the number of type-labeled vertices in R multiplied by the type–deletion penalty from the similarity matrix. If R_1 and R_2 are both empty, then $H(R_1, R_2) = 0$ and R_1', R_2', and their isomorphism f are empty as well. If either R_1 or R_2 (say R_1) is a deletion with two empty subtrees and their incident edges being empty as well, while the other (R_2) is nonempty, then $H(R_1, R_2) = k(R_2)$, R_1' is a tree of deletions topologically isomorphic to R_2, and $R_2' = R_2$. By induction, in the forward part, only qualities are computed and a link is given to the transformation at which their maximum is reached (see the induction step below), while the isomorphic extensions T_1' and T_2' are constructed in the backward part.

Induction step. Let $\{R_1, R_2\}$ be the current pair of nonempty trees in P. The algorithm computes the quality $H(R_1, R_2)$ of each of the seven *transformations* $(R_1, R_2) \to (R_1', R_2')$ listed in Fig. 2, and creates a *link* to that of them (including its arguments) at which the quality is maximal as long as $R_1 < T_1$ or $R_2 < T_2$. Then, by backward induction from the pair (T_1, T_2) to the induction base, the algorithm computes the trees $R_1' = T_1'$ and $R_2' = T_2'$ and, if necessary, their isomorphism f.

Let x and y be the *labels* of the roots in R_1 and R_2. In Fig. 2, the current trees $\{R_1, R_2\}$ are shown in the upper panels, and their transformations $\{R_1', R_2'\}$, in the lower panels. *Denote* by c_{xy} the similarity of the labels x and y in the similarity matrix, including penalties. The result $(\cdot)'$ is determined by induction by the indicated linear order; The root of the tree $(S^-)'$ is labeled by deletion.

(a) In R_1 and R_2, the child subtrees are A and B, C and D from the roots in R_1 and R_2, respectively (see Fig. 2a); the subtrees A, B, C, D are all empty or all nonempty (note that the case when, for example, A and B are empty, while C and D are not is reduced to case (b) below by applying operation 3 to x). Let $H(\{R_1, R_2\}) = H(\{A, C\}) + H(\{B, D\}) + c_{xy}$. Similarly, we compute $H(\{A, D\}) + H(\{B, C\}) + c_{xy}$ and, finally, $H(\{R_1, R_2\})$ is set equal to the maximum of these two variants. The *link* indicates the transformation (out of seven ones) and its arguments at which the maximum is reached. By induction, $\{A', C'\}$ and $\{B', D'\}$ are known, and they are attached as shown in Fig. 2a. The isomorphism f: $R_1' \to R_2'$ is equal to the union of the isomorphisms f_1 and f_2 supplemented with the correspondence $x \to y$. These data are not used in the forward part. All vertices of T_1 are contained in T_1' with the same labels. At vertices of T_1' below T_1-leaves and above the T_1-root or incomparably with it, there are only deletions; similarly, for T_2 and all seven transformations. This is used in the proof of Theorem 1.

(b) For the trees R_1 and R_2, we set $A = R_1$ and B and C are child subtrees in R_2 from its root y (see Fig. 2b); the subtrees A, B, and C are nonempty. Let $H(\{R_1, R_2\}) = H(\{A^-, B\}) + c_{xy} + k(C)$. Similarly, we compute $H(\{A^-, C\}) + c_{xy} + k(B)$ and two variants of the quality $H(\{R_2, R_1\})$. Here, we use the following *notation*: $\{R_2, R_1\}$ is a pair, where R_2 is a new A attached to y and the union of the previous B and C, while R_1 is the previous A with the root x divided into the new child subtrees B and C. Finally, $H(\{R_1, R_2\})$ is set equal to the maximum of these four variants. Here, operation 3 is applied to x and $(A^-)'$ and $-C$ are attached to x, while B' and C are attached to y. The isomorphism f: $R_1' \to R_2'$ is defined as described above.

(c) For R_1 and R_2, we set $A = R_1$ and $B = R_2$ (see Fig. 2c); the trees A and B are nonempty. Let $H(\{R_1, R_2\}) = c_{xy} + k(A^-) + k(B^-)$. Here, operation 3 is applied to x and y and A^-, $-B$, $-A$, and B^- are attached to them. The isomorphism f: $R_1' \to R_2'$ is trivial.

(d) Assume that at least one of the labels x and y is not a deletion. For the trees R_1 and R_2, we set $A = R_1$ and $B = R_2$ (see Fig. 2d); the trees A and B are nonempty. Let $H(\{R_1, R_2\}) = H(\{A^-, B^-\}) + c_{xy}$. Here, operation 3 is applied to x and y and $(A^-)'$ and $(B^-)'$ are attached to them. The isomorphism f: $R_1' \to R_2'$ is defined by induction.

(e) For the trees R_1 and R_2, we consider the child subtrees A and B from the root x and set $C = R_2$ with the root y (see Fig. 2e); the subtrees A, B, and C are nonempty. Let $H(\{R_1, R_2\}) = H(\{A, C\}) + c_{x-} + k(B)$. Similarly, we compute $H(\{B, C\}) + c_{x-} + k(A)$ and two variants for $H(\{R_2, R_1\})$. Here, as in (b), we use the following notation: $\{R_2, R_1\}$ is a pair in which R_2 is the previous C with the root y divided into new child subtrees A and B, while R_1 is a new C attached to x and the union of the previous A and B. Here, operation 2 is applied to y and A', B, C', and $-C$ are attached to y. Finally, $H(\{R_1, R_2\})$ is set equal to the maximum of these four variants. The isomorphism f: $R_1' \to R_2'$ is defined by induction.

(f) Assume that x is not a deletion. For the trees R_1 and R_2, we set $A = R_1$ and $B = R_2$ (see Fig. 2f); the trees A and B are nonempty. Let $H(\{R_1, R_2\}) = H(\{A^-, B\}) + c_{x-}$. Similarly, we compute $H(\{A, B^-\}) + c_{x-}$, and, finally, $H(\{R_1, R_2\})$ is set equal to the maximum of these two variants. Here, operation 3 is applied to x, while operation 2 is applied to y and $(A^-)'$ and B' are attached to y. The isomorphism f: $R_1' \to R_2'$ is defined by induction.

(g) For the trees R_1 and R_2, we set $A = R_1$ and $B = R_2$ (see Fig. 2g); the trees A and B are nonempty. Let $H(\{R_1, R_2\}) = k(A) + k(B)$. Here, operation 3 is applied to x, while operation 2 is applied to y and A^-, $-B$, $-A$, and B are attached to it. The isomorphism f: $R_1' \to R_2'$ is trivial.

After the forward part of the algorithm is completed, in the forward pass starting from $\{T_1 = R_1, T_2 = R_2\}$, the algorithm forms $T_1' = R_1'$ and $T_2' = R_2'$ according to the links placed in the forward part. □

Remark 1. Connected parts of the original trees (not necessarily subtrees) are aligned as follows. In the forward pass, any negative quality of a pair $p \in P$ is replaced by zero. After the backward pass, all f-isomorphic subtrees with a quality of 0 are deleted from the resulting isomorphism f, and the quality of the restriction of f to the connected parts of the original trees is equal to the original quality.

4. ACCURACY OF THE ALGORITHM AND EXAMPLE OF ITS WORK

Theorem 1. *The algorithm produces alignments T_1' and T_2' of two trees T_1 and T_2 with an isomorphism of maximum quality. The trees T_1' and T_2' are isomorphic extensions of T_1 and T_2. The running time of the algorithm is quadratic in the size of the initial data.*

Proof sketch. Using induction on pairs $\{R_1, R_2\}$, we show that the algorithm constructs an isomorphism f_0: $R_1' \to R_2'$ of maximum quality. Let f: $R_1'' \to R_2''$ be an isomorphism of maximum quality and G_1, G_2 be sequences of operations extending R_1 to R_1'' and R_2 to R_2''. Let us show that $H(f) = H(f_0)$. If either R_1 or R_2 is a deletion with two empty subtrees, then the equality is obvious. Let r_1 and r_2 be the roots in R_1 and R_2 and vertices in R_1'' and R_2'' for which $f(r_1) = r_2$. The subtrees with roots r_1 and r_2 are isomorphic. Denote them again by r_1 and r_2, respectively. The complements of the subtrees $R_1''\backslash r_1$ and $R_2''\backslash r_2$, which consist of only deletions, are isomorphic as well. By convention, a deletion—deletion correspondence has a penalty of 0, so $H(f) = H(f{\restriction}r_1)$; the restriction $f{\restriction}r_1$ is again denoted by f. Let u_1 and u_2, u_3 and u_4 denote the roots of child subtrees from the vertices r_1 and r_2 and simultaneously these subtrees themselves in R_1'' and R_2''. In view the isomorphism, $f(u_1) = u_3$ and $f(u_2) = u_4$ or vice versa (say the former case occurs). We obtain isomorphisms f_1: $R_1'' \restriction u_1 \to R_2'' \restriction u_3$ and f_2: $R_1'' \restriction u_2 \to R_2'' \restriction u_4$, i.e., f is the union of f_1 and f_2 together with $r_1 \to r_2$.

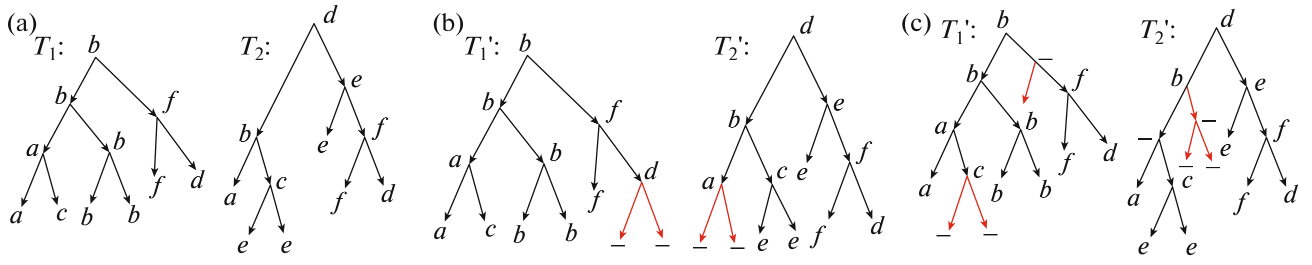

Fig. 3. (a) Initial trees T_1 and T_2. (b) Similarity of types is 1 and penalty is –1. Deletions and (red) edges are added, after which topologically isomorphic trees T_1' and T_2' are obtained. (c) Similarity of a type to itself is 4 and to another type is –3; penalty for a deletion is –2. Deletions and (red) edges are added, after which topologically isomorphic trees T_1' and T_2' are obtained.

Let $v_i = G_1^{-1}(u_i)$ and $v_i = G_2^{-1}(u_i)$ (for i = 1 or 2 and for i = 3 or 4). Let S_i be subtrees in R_1 and R_2 from v_i. The sequence of operations G_i transforms S_i into the tree $R_1'' \upharpoonright u_i$. Therefore, pairs $\{S_1, S_3\}$ and $\{S_2, S_4\}$ that are strictly less than $\{R_1, R_2\}$ generate $\{R_1'' \upharpoonright u_1, R_2'' \upharpoonright u_3\}$ and $\{R_1'' \upharpoonright u_2, R_2'' \upharpoonright u_4\}$ using the same G_i. The isomorphisms f_1: $R_1'' | u_1 \to R_2'' | u_3$ and f_2: $R_1'' | u_2 \to R_2'' | u_4$ are of maximum quality; otherwise, S_1 and S_3 can be replaced by other subtrees that generate isomorphic subtrees of higher quality instead of $R_1'' \upharpoonright u_1$ and $R_2'' \upharpoonright u_3$; S_2 and S_4 are replaced in a similar fashion. By the induction hypothesis, $H(f_1)$ and $H(f_2)$ are equal to the qualities of the isomorphisms produced by the algorithm on the pairs $\{R_1'|u_1, R_2'|u_3\}$ and $\{R_1'|u_2, R_2'|u_4\}$, whence $H(f) = H(f_0)$.

Assume that $f(r_1) = d \neq r_2$. The following three cases are possible.

(1) The vertex d is above r_2 and, hence, is a deletion. Once again, u_i denotes child vertices from r_1 and d. Then d in R_2'' has exactly one child subtree D (say with the root u_4), which a tree of deletions isomorphic to the child subtree R from r_1 in R_2''. The quality of this isomorphism f_2 is $k(R)$. Let u_1 be a child vertex from r_1 not lying in R, and let u_3 be a child vertex from d not lying in D. The trees from u_1 and u_3 are f_1-isomorphic, and f is the union of f_1 and f_2 together with $r_1 \to d$. Repeating the arguments for the case $f(r_1) = r_2$ in the case of u_1, we obtain the corresponding S_1 and $S_3 = R_2$. Here, $\{S_1, S_3\}$ is strictly less than $\{R_1, R_2\}$.

(2) The vertex d is incomparable with r_2 and, hence, is a deletion. Then the isomorphism f maps all vertices from R_1 to deletions and $H(f) = k(R_1) + k(R_2)$, as for transformation (g) in Fig. 2g. Since $H(f_0) \geq k(R_1) + k(R_2)$, we obtain $H(f) = H(f_0)$.

(3) The vertex d is below r_2. The isomorphism f preserves the order relation on the tree, so $f^{-1}(r_2)$ is a deletion lying above r_1. This case is symmetric to case (1).

Thus, $H(f) = H(f_0)$, i.e., the algorithm is exact. Its running time is quadratic in the size of the initial trees, because so is the number of pairs in P and the processing of each pair takes constant time. □

Example 1. Consider the trees T_1 and T_2 shown in Fig. 3a, whose vertices are labeled by types a, b, c, d, e, and f (empty subtrees in leaves are not shown). The elements of the similarity matrix are equal to 1 (type—type) and to –1 (type—deletion). The results T_1' and T_2' produced by the algorithm are shown in Fig. 3b. The quality of the isomorphism is 9 – 4 = 5.

Example 2. Consider the same trees T_1 and T_2. Similarity of a type to itself is 4 and to another type is –3; penalty for type—deletion (and vice versa) is –2. The backward pass of the algorithm begins with the pair $\{T_1 = R_1, T_2 = R_2\}$ and the (previously obtained) link *a1* (subtrees A and C, B and D). At the second step, for the first obtained pair $\{R_1 = A, R_2 = C\}$, there was link *b3* to the new pair $\{R_1, R_2\}$, where $R_1 = C$ (new A) and R_2 is a tree with the root a and leaves a and c (new child B). For the second pair $\{R_1 = B, R_2 = D\}$ of the first step, there was link *e4* to the new pair $\{R_1, R_2\}$, where R_1 is a tree with the root f and leaves f and d (new child B) and $R_2 = B$ (new C), which then coincides with R_1. The next step of the algorithm is trivial: there was link *a1* everywhere. The result produced by the algorithm is shown in Fig. 3c: alignment (according to the isomorphism) has six identical types, one unequal type, and eight type—deletion correspondences. The quality of the isomorphism is 24 – 3 – 16 = 5. □

Remark 2. For polytomic initial trees, the algorithm is easy to modify so that it will output a pair of their binary polytomy resolutions of maximum quality. For a pair of vertices $\{x, y\}$ from T_1 and T_2, respectively (ordered from the leaves to the root) we consider a nonempty set X of child edges for x and Y for y. The set X *generates* a subtree in T_1 that lies below the edges from X; if $|X| > 1$, then these edges with their upper ends are included as well. For Y, the situation is similar. By a simplification of X we mean a simplification of this subtree. If X or Y is a singleton, then the subtrees generated by X and Y have binary resolutions of maximum quality, which are known by induction. Otherwise, these two sets are divided into nonempty subsets M_1 and M_2, M_3 and M_4. Consider the restrictions of T_1 and T_2 from the vertices x and y, respectively, to M_1 and M_2, M_3 and M_4. The algorithm searches through pairs of sets X, Y and their simplifications in ascending order of cardinality of X, and, for X of fixed cardinality, in ascending order of cardinality of Y. For fixed X and Y, the search is performed in the following order: (X^-, Y^-), (X^-, Y), (X, Y^-), and (X, Y). For each pair (X, Y), all pairs of their partitions are searched through in any order; the other three pairs are considered in a similar manner. Let X^* denote X or X^-. By induction, we know binary resolutions of maximum quality for pairs of restrictions with roots x and y on (M_1, M_3), (M_1, M_4), (M_2, M_3), (M_2, M_4), (M_1, Y^*), (M_2, Y^*), (X^*, M_3), (X^*, M_4), (X^-, Y^*), and (X^*, Y^-), respectively. From all pairs of partitions, for each of them searching through the transformations described in the algorithm (Fig. 2), we choose a partition of maximum quality and obtain a pair of binary resolutions of maximum quality for the subtrees generated by X and Y. Taking the largest X and Y for $\{x, y\}$, we obtain a binary resolution of maximum quality for subtrees from x and y, thus reaching the pair of roots of the original trees T_1 and T_2.

Here, quadruples of subsets are searched through, which yields the upper bound 2^{4k} for the time of processing of a single vertex pair, where k is the maximum number of child vertices for a tree vertex. The quadratic running time of the algorithm in the binary case is multiplied by a number of order 2^{4k}.

The problem of transforming polytomic ordered trees (in our case, trees are unordered) has been extensively studied. Numerous references can be found, for example, in [5]. For this problem, there is an exact solution algorithm with cubic running time. Under a certain assumption, it was shown in [5] that this time cannot be significantly improved.

FUNDING

This work was supported by the Russian Science Foundation, project no. 24-44-00099, https://rscf.ru/en/project/24-44-00099/.

CONFLICT OF INTEREST

The authors of this work declare that they have no conflicts of interest.

REFERENCES

1. D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology* (Cambridge Univ. Press, Cambridge, 1997).
2. K. Yu. Gorbunov and V. A. Lyubetsky, "An almost exact linear algorithm for transformation of chain-cycle graphs with optimization of the sum of operation costs," Dokl. Math. **102** (2), 376–379 (2020). https://doi.org/10.1134/S1064562420050324
3. M. Yuan, X. Yang, J. Lin, X. Cao, F. Chen, X. Zhang, Z. Li, G. Zheng, X. Wang, X. Chen, and J.-R. Yang, "Alignment of cell lineage trees elucidates genetic programs for the development and evolution of cell types," iScience **23**, 101273 (2020). https://doi.org/10.1016/j.isci.2020.101273
4. https://github.com/Chenjy0212/mdelta. Accessed January 20, 2024.
5. K. Bringmann, P. Gawrychowski, Sh. Mozes, and O. Weimann, "Tree edit distance cannot be computed in strongly subcubic time (unless APSP can)," ACM Trans. Algorithms **16** (4), 48 (2020). https://doi.org/10.1145/3381878

Translated by I. Ruzanova