An Algorithm for Comparing Similarity Between Two Trees: Edit Distance with Gaps[*]

Hangjun Xu [†]

April 7, 2014

Contents

1 Introduction: 1 minute

Welcome to my defense presentation. The title of my talk is: **An Algorithm for Comparing Similarity Between Two Trees**. What I did in this project is that I computed the **general gap edit distance between two binary trees in $O(n^5)$ time**. Prior to our work, no such distance computation was given since the computation for general gap edit distance between arbitrary tree case has been proved to be NP-hard. Before I could tell you all about that, I need to tell you what tree edit distance is, and what gap is. This talk consists of three parts: First, I will provide some background in edit distance, tree comparisons, and I will discuss the classic tree edit distance algorithm. Second, I will talk about two different models of gaps in trees.

[*]M.S. defense presentation. Each ● represents a slide.

[†]Ph.D. Candidate, Department of Mathematics, Duke University. hangjun@math.duke.edu, homepage: `http://fds.duke.edu/db/aas/math/hangjun`.

I will present the algorithms for computing the edit distance with each of these gap models. Finally, I will discuss an application of edit distance with gaps in terrain comparison. I will close this talk with some open problems and future works.

2 Shape Comparison using Edit Distance: 15 minutes

• Shape matching concerns that problem of **quantifying and computing similarity between two shapes** like point sets, curves, and surfaces. Shape matching is also closely related to **clustering** and classification of various shapes.

2.1 String Matching

• The **edit distance**, or Levenshtein distance, was first used to quantify similarity between two sequences of characters, also called **strings**. The way such similarity is defined is to transform one string to anther via **edit operations** include insertion, deletion and substitution. Each operation has a cost, and the edit distance is defined to be the minimal cost of such transformation. Edit operations give rise to **alignments** of strings. An alignment is a way of placing one string on top of another to get a matching between the characters in each string. A deleted character is aligned with a special **blank character**. For instance, the string **save** can be aligned with the string **salvage** with three deletions as:

```
s a - v - - e
s a l v a g e
```

The edit distance is the same as the optimal string alignment (i.e. with minimal cost).

• Applications of string matching include comparisons of DNA sequences, which consists of the characters **A, G, C, T**; spell checking; and string searchings.

• Now given two strings S_1 and S_2 with m and n characters, respectively. A **gap** in an alignment of S_1 and S_2 is a **longest consecutive blank characters**. Different gap penalty functions are used in different applications, and the algorithms for computing the optimal alignment depend on the choice of gap penalty functions. For instance:

(1) Linear gap penalty (each blank character in a gap is charged equally): $O(mn)$;

(2) Convex gap penalty (the first blank character in a gap is charged more than the others): $O(mn)$;

(3) Arbitrary gap penalty: $O(mn^2 + m^2n)$.

The above running times are all based on dynamic programming algorithms. There are many parallel algorithms developed that can improve these running times.

• A related problem to string matching is the problem of **trajectory comparisons**. A trajectory is simply a curve in $\mathbb{R}^2$. Sampling of a trajectory gives us a sequence of points in the plane. We can use the similarity of two such sequences as a measure of similarity between the two trajectories. S. Sankararman et al [17] defined a comparison model that combines string alignment and dynamic time warping. We simply remark that string alignment is good at dealing with noise since it can skip points in an alignment (i.e. gaps). However, since a point can only be aligned another point, it is not ideal if the sample rates are not uniform. On the other hand, dynamic time warping (i.e. average Fréchet distance) allows multiple points to be aligned with a single points, which remedies the disadvantage of string alignment.

2.2 Classic Tree Edit Distance

• The central problem we study in this project is **how to quantify similarity between two trees**. The motivation comes from the fact that many complicated nonlinear shapes have underlying linear tree structures. Moreover, these tree structures often preserve many topological and geometric properties, e.g. *connectedness, genus, homotopy type, critical points*, etc. Therefore, instead of comparing the rather complicated shapes, we can simply compare their underlying tree structures. **Let me give you two examples to illustrate this**.

• Consider a region S in the plane with boundary C a planer curve. The **medial axis** of S is defined to the collection of centers of disks contained in S that intersect C tangentially at least **twice**. If C is a polygon, then the medial axis is a tree. One can picture the medial axis as obtained by a deformation retract of S. Given the medial axis and all the radii of the disks, one can **reconstruct** S in many cases.

• The second example is the contour tree of a terrain. Here is a picture of a terrain in $\mathbb{R}^3$ (point to the picture), which is a graph of a height function defined on $\mathbb{R}^2$.

Notice that the water ponds in Figure 1 is the level set of the height function with respect to some z value. A connected component of a level set is called a **contour**. One key observation is that the level sets only change topology at **critical points** of the height functions, which consists of local maximum, local minimum and saddles. A contour is first born at a local minimum, and dies at local maximum. At a saddle point, either two contours join and become a single contour, or a single contour splits into two. **The contour tree is defined to be a graph whose vertices are critical points, and (u, v) is an edge if there is a contour that appears at v**

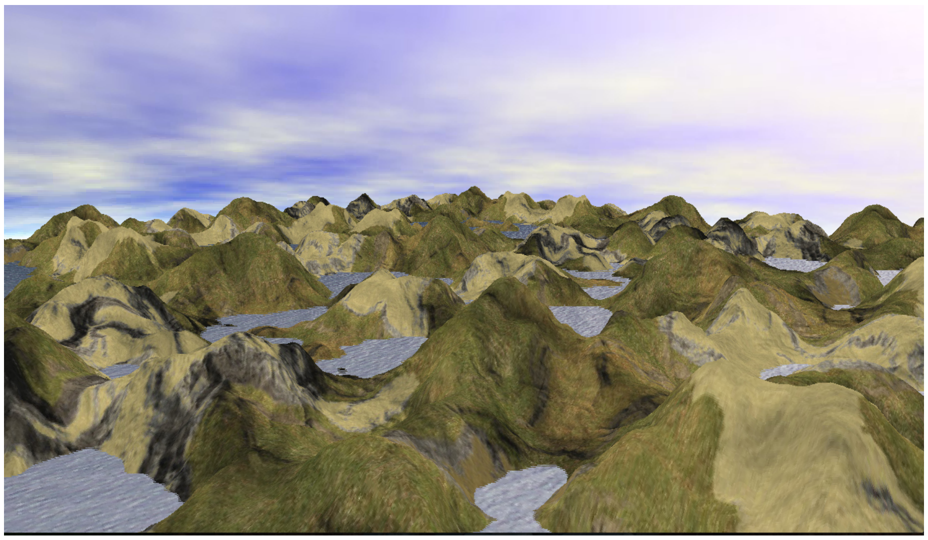

Figure 1: Animated terrain drawn using polynomial height functions. Picture by Tingran Gao and Hangjun Xu.

and disappears at u. It was shown that this actually defines a **tree**. Notice that no matter how complicated the terrain is, its contour tree is always a tree, and captures the evolution of contours of that terrain. Thus, we can use similarity between two contour trees as a measure of similarity between the two corresponding terrains.

• So now the question is: **How do we measure similarity between two trees**? A well-known distance function, called **edit distance**, is motivated by the edit operations in string matching. For the rest of the talk, we assume that all trees are **ordered** and **labeled**, where ordered means that an ordering is defined among all the siblings; and labeled means that each node has assigned a symbol from a finite alphabet Σ. Similar to the string matching case, we have three edit operations (see Figure 2):

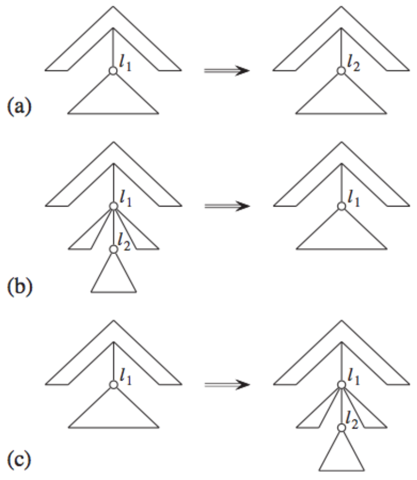

Figure 2: Edit operations. Picture from [5]

(a) Rename. To rename label on one node to another.

(b) Delete. To delete a node u, and all children of u become children of the parent of u, while maintaining the order.

(c) Insert. To insert a node u as a child of u'. A consecutive sequence of children of u' now becomes the children of u.

Now for the cost of each edit operation, we choose a metric

$$p : \Sigma \cup \{\perp\} \times \Sigma \cup \{\perp\} \longrightarrow \mathbb{R}^{\geq 0},$$

symmetric, positive definite and satisfies the triangle inequality. Here $p(u, v)$ is the cost of relabeling u to v, and $p(*, \perp)$ or $p(\perp, *)$ denote the cost of deletion or insertion. $\perp$ plays the same role as the **blank character** in string alignment.

An edit script $S := \{s_1, s_2, \cdots, s_n\}$ taking T_1 to T_2 is simply a sequence of edit operations s_i. The cost of S is defined to be the total costs of all the edit operations (simply point to the formula in the slides).

Now the **edit distance between T_1 and T_2** is defined to be the minimal cost edit script taking T_1 to T_2 (simply point to the definition in the slides).

• Here is a graphic representation of an edit script, which is called a **mapping** from T_1 to T_2, analogues to an alignment between two strings (see Figure 3). You

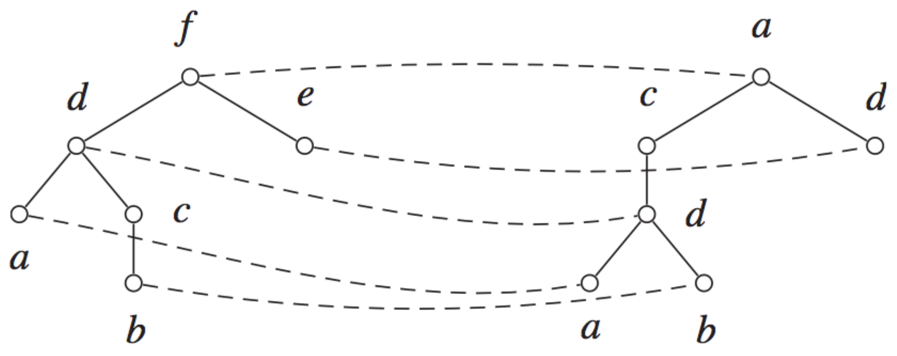

Figure 3: A mapping between two trees. Picture from [5].

can see that nodes labeled c on both trees are deleted; node labeled f on the left is relabeled to a; and the node labeled e on the left is relabeled to d.

More formally, we have (Here, simply say that a mapping M is map from T_1 to T_2 that is one-to-one, and preserves the sibling order and the ancestor order. No need to read through the definitions.):

Definition 2.1 (Mapping Between Two Trees). *$M \subset V_1 \times V_2$. For any (u, v) and (u', v') in M:*

(1) $u = u'$ if and only if $v = v'$; this is called the **one-to-one condition***;*

(2) u is to the left of u' if and only if v is to the left of v'; this is called the **sibling order condition***;*

(3) u is an ancestor of u' if and only if v is an ancestor of v'; this is called the **ancestor order condition***.*

• **The edit distance is the same as an optimal mapping**. There are many studies on computing the edit distance efficiently. We now list some representative works. Let $m := |T_1|$, $n := |T_2|$, $D_i := depth(T_i)$ and $L_i := |leaves(T_i)|$, $i = 1, 2$.

(1) Straightforward dynamic programming algorithm: $O(m^2 n^2)$;

(2) Zhang and Shasha [24], 1989: $O(mn \min\{D_1, L_1\} \min\{D_2, L_2\})$, using **key roots**;

(3) Klein [16], 1998: $O(m^2 n \log n)$, using **path decomposition**;

(4) Chen [8], 2001: $O(mn + L_1^2 n + L_1^{2.5} L_2)$;

(5) Dulucq, Touzet [13], 2003: $O(mn \log^2 n)$;

(6) Demaine et al [10], 2009: $O(m^2 n)$.

• We briefly present the straightforward dynamic programming algorithm, which serves as a motivation for our work. Here are some terminologies (Simply point to the slides, and read through them quickly):

(1) Fix left-to-right **postorder** traversal $\longrightarrow$ numbering among all the nodes.

(2) $T[i]$: the i^{th} node.

(3) $l(i)$: the index of the leftmost leaf descendant of the subtree rooted at $T[i]$. (e.g. $T[i]$ is a leaf $\Longrightarrow l(i) = i$).

(4) $r(i)$: the index of the leftmost leaf descendant of the subtree rooted at $T[i]$. (e.g. $T[i]$ is a leaf $\Longrightarrow r(i) = i$).

(5) $p(i)$: parent of $T[i]$; $anc(i)$: and $desc(i)$: descendants of $T[i]$.

(6) $Q[i..i', j..j']$: edit distance between $T[i..i']$ and $T[j..j']$.

• We have the following recurrence for Q:

Theorem 2.1. *For any $i \in desc(i_1)$ and $j \in desc(j_1)$,*

$$Q[l(i_1)..i, l(j_1)..j] = \min \begin{cases} Q[l(i_1)..i-1, l(j_1)..j] + p(i, \perp) \\ Q[l(i_1)..i, l(j_1)..j-1] + p(\perp, j) \\ Q[l(i_1)..l(i)-1, l(j_1)..l(j)-1] + Q[l(i)..i-1, l(j)..j-1] + p(i,j) \end{cases}$$

To see this, simply note that either i is deleted (**point to** $p(i, \perp)$), in which case we have the first recursion; or j is deleted (**point to** $p(\perp, j)$), in which case we have the second recursion; or i is mapped to j (**point to** $p(i,j)$ **and Figure 4**), in which case $T_1[l(i_1)..l(i)-1]$ is mapped with $T_2[l(j_1)..l(j)-1]$, and the remaining $T_1[l(i)..i-1]$ is mapped with $T_2[l(j)..j-1]$. This proves Theorem 2.1.

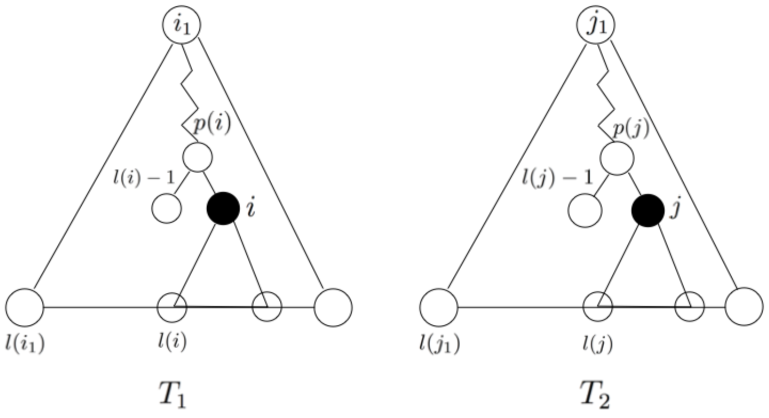

Figure 4: Third case in classic edit distance recurrence.

3 Here is What I did: Tree Edit Distance with Gaps: 20 minutes

• **Now, let me tell about what I did for this project**. Recall that in string alignment, we have the notion of a **gap** that denotes a longest consecutive sequence of blank characters. We see from the trajectory matching example, these gaps can be viewed as noise in the input. Two natural questions are: **How do we define gaps in trees**? and **What should the gap penalty function be**? One heuristic is that a single "large" gap is more likely to appear than several isolated smaller gaps. This suggests that we use a **convex function** as gap penalty (point to the definition of convexity in the slides). In particular, an affine function (point to the affine function in the slides) is convex. **For the following, we always assume that the gap penalty is affine**.

3.1 Complete Subtree Gap Model

• The first model of gap is called the **complete subtree model**, first defined by Touzet:

Definition 3.1 (Complete Subtree Gap Model, Touzet, [22]). *Given a tree T with vertex set V. A gap g_v of T is the complete subtree rooted at some vertex $v \in V$.*

The edit operations now become:

1. Relabel a node;
2. Delete a gap;
3. Insert a gap.

Notice that in this model, if one node is deleted, all of its descendants will be deleted since deletion and insertion are done in gaps. Given a mapping M, its cost is given by

$$\gamma(M) := \sum_{(u,v)\in M} p(u,v) + \sum_{g\in G} a + b|g|, \quad u \in V_1 \ v \in V_2, \tag{3.1}$$

This is the function that we are trying to minimize.

• Since we are using affine gap penalty, starting a gap has different penalty than continuing a gap. To determine whether a deleted node is starting or continuing a gap, we need the information about its parent. This suggest that we use the preorder traversal or order the nodes. We define three functions. For $1 \le i' \le i \le m := |T_1|$, and $1 \le j' \le j \le n := |T_2|$, set:

$$\begin{cases} Q[i'..i, j'..j] := \gamma(T_1[i'..i], T_2[j'..j]); \\ Q_{\perp *}[i'..i, j'..j] := \gamma(T_1[i'..i], T_2[j'..j]) \text{ such that } i \to \perp; \\ Q_{* \perp}[i'..i, j'..j] := \gamma(T_1[i'..i], T_2[j'..j]) \text{ such that } \perp \to j \end{cases} \tag{3.2}$$

where m is the number of nodes in T_1, and n is the number of nodes in T_2. **Thus, our goal is to compute $Q[1..m, 1..n]$.**

• We can set the boundary conditions of $Q, Q_{\perp *}$ and $Q_{* \perp}$ as follows (here $\emptyset$ means an empty tree):

$$\begin{aligned} Q[\emptyset, \emptyset] &= 0, \\ Q[1..i, \emptyset] &= \infty, && \text{(for } 1 \le i \le m) \\ Q[\emptyset, 1..j] &= \infty, && \text{(for } 1 \le j \le n) \end{aligned}$$

$$Q_{\perp *}[1..i, \emptyset] = \infty, \qquad \text{(for } 1 \le i \le m)$$
$$Q_{\perp *}[\emptyset..1..j] = aj + b, \qquad \text{(for } 1 \le j \le n)$$

since it is impossible to match $T_1[1..i]$ to an empty tree such that i is a gap point; and there is a unique matching between en empty T_1 with $T_2[1..j]$: we have j gap points.

By symmetry, we set

$$Q_{* \perp}[1..i, \emptyset] = ai + b, \qquad \text{(for } 1 \le i \le m)$$
$$Q_{* \perp}[\emptyset, 1..j] = \infty, \qquad \text{(for } 1 \le j \le n)$$

• The recurrences for $Q, Q_{\perp *}$ and $Q_{* \perp}$ are given by, with i, j descendants of i_1 and j_1 respectively:

Theorem 3.1 (Recurrence of Auxiliary Matrices in Complete Subtree Gap Model)**.** *Given the preorder ordering on the nodes of two ordered labeled trees T_1 and T_2. Fix nodes $i_1 \in V_1, j_1 \in V_2$. For any $i \in desc(i_1)$ and $j \in desc(j_1)$, we have the following recurrence relations:*

$$Q[i_1..i, j_1..j] = \min \begin{cases} Q[i_1..i-1, j_1..j-1] + p(i,j) \\ Q_{\perp *}[l_1..i, j_1..j] \\ Q_{* \perp}[i_1..i, j_1..j] \end{cases} \tag{3.3}$$

$$Q_{\perp *}[i_1..i, j_1..j] = \min \begin{cases} Q[i_1..i-1, j_1..j] + (a+b) \\ Q_{\perp *}[i_1..p(i), j_1..j] + b(i - p(i)) \end{cases} \tag{3.4}$$

$$Q_{* \perp}[i_1..i, j_1..j] = \min \begin{cases} Q[i_1..i, j_1..j-1] + (a+b) \\ Q_{* \perp}[i_1..i, j_1..p(j)] + b(j - p(j)) \end{cases} \tag{3.5}$$

Here $p(i)$ (resp. $p(j)$) is the index of parent node (if exists) of i (resp. j).

• Let's only look at the recurrence for $Q_{\perp *}$. If $p(i)$ is not a gap node or does not exit, then i is starting a new gap, hence we have the firs recursion; Otherwise $p(i)$ is a gap node, and i is continuing a gap (point to Figure 5).

All the descendants of $p(i)$ should all be deleted. There are $i - p(i)$ many of them, and hence the penalty $b(i - p(i))$, and we backtrack this recursion to ending at $p(i)$.

• Here is the algorithm for computing the recurrences (point to the algorithm in the slides). For each i_1, j_1, i_1 going from 1 to m; j_1 going from 1 to n, we compute $Q[i_1..i, j_1..j]$ in increasing order of i and j; and for each fixed i, j, we compute Q by first compute $Q_{\perp *}$, then compute $Q_{* \perp}$. The running time is $O(m^2 n^2)$.

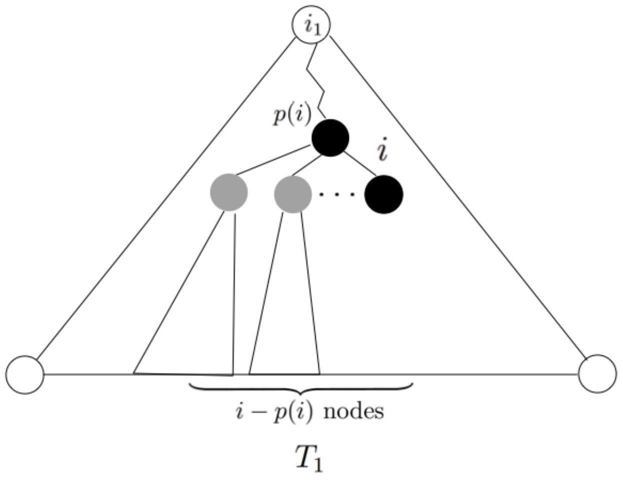

Figure 5: $p(i)$ and i are both gap nodes.

3.2 General Gap Model

• The complete subtree model for gaps is too restrictive. We now study a general gap model, first defined by Touzet:

Definition 3.2 (General Gaps Model, Touzet [22])**.** *Given an ordered labeled tree T with vertex set V and edge set E. A gap g is a tree with vertex set a subset of V and edges in E whose both end points lie in that subset.*

That is, a gap is any part of T that is a tree on its own. The edit operations are now modified to: dmissible Edit Operations:

1. Relabel a node;
2. Delete a gap: descendants of a gap will become children of the parent of the root of the gap;
3. Insert a gap.

Unfortunately, Touzet showed that **the computation of general gap edit distance is NP-hard even for affine gap penalty functions**. This suggests that maybe we should **restrict our comparison to a subclass of trees in order to compute this distance in polynomial time**.

• Thus, we compute the general gap edit distance for binary trees. Here are the recurrences of $Q, Q_{\perp *}$ and $Q_{* \perp}$, defined in the same way as in the complete subtree case, with i, j descendants of i_1 and j_1 respectively.

Theorem 3.2 (Recurrence of Auxiliary Matrices in General Gap Model for Binary Trees)**.** *Given the preorder ordering on the nodes of two ordered labeled trees T_1 and*

T_2. Fix nodes $i_1 \in V_1, j_1 \in V_2$. For any $i \in desc(i_1)$ and $j \in desc(j_1)$, we have the following recurrence relations:

$$Q[i_1..i, j_1..j] = \min \begin{cases} Q[i_1..i-1, j_1..j-1] + p(i,j) \\ Q_{\perp *}[i_1..i, j_1..j] \\ Q_{* \perp}[i_1..i, j_1..j] \end{cases} \tag{3.6}$$

$$Q_{\perp *}[i_1..i, j_1..j] = \min \begin{cases} Q[i_1..i-1, j_1..j] + (a+b) \\ Q_{\perp *}[i_1..i-1, j_1..j] + b \\ \min\limits_{j_1 \le k \le j} \big\{ Q_{\perp *}[i_1..p(i), j_1..k] \\ \qquad + Q[p(i)+1..i-1, k+1..j] + b \big\} \end{cases} \tag{3.7}$$

$$Q_{* \perp}[i_1..i, j_1..j] = \min \begin{cases} Q[i_1..i, j_1..j-1] + (a+b) \\ Q_{* \perp}[i_1..i, j_1..j-1] + b \\ \min\limits_{i_1 \le k \le i} \big\{ Q_{* \perp}[i_1..k, j_1..p(j)] \\ \qquad + Q(k+1..i, p(j)+1..j-1) + b \big\} \end{cases} \tag{3.8}$$

Here $p(i)$ (resp. $p(j)$) is the index of parent node (if exists) of i (resp. j).

• We only sketch the recurrence for $Q_{\perp *}$. If $p(i)$ is not a gap node, then i is starting a new gap, hence the first recursion. If $p(i)$ is a gap node and i is its left child (point to Figure 6)

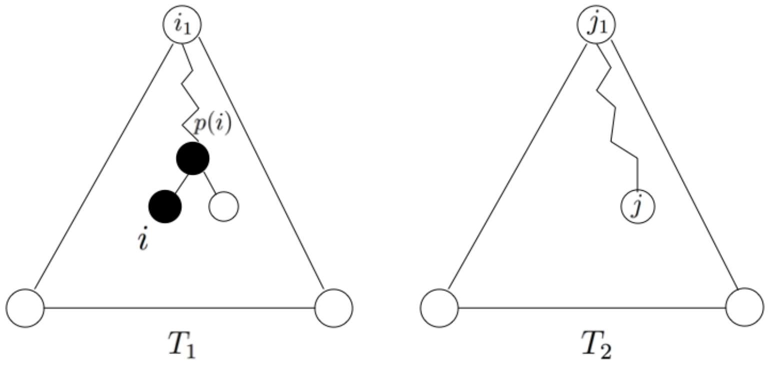

Figure 6: $p(i)$ is a gap node and i is its left child. Gap nodes are labeled black.

Then i is continuing a gap and we backtrack to ending at $i-1$.

• Otherwise i is the right child (point to Figure 7) Then i is also continuing a gap, and $T_1[i_1..p(i)]$ can be mapped with $T_2[j_1..k]$, for any $j_1 \le k \le j$, and we backtrack to ending at $p(i)$. The remaining part of T_1: $T_1[p(i)+1..i-1]$ is mapped with the remaining part of T_2: $T_2[k+1..j]$.

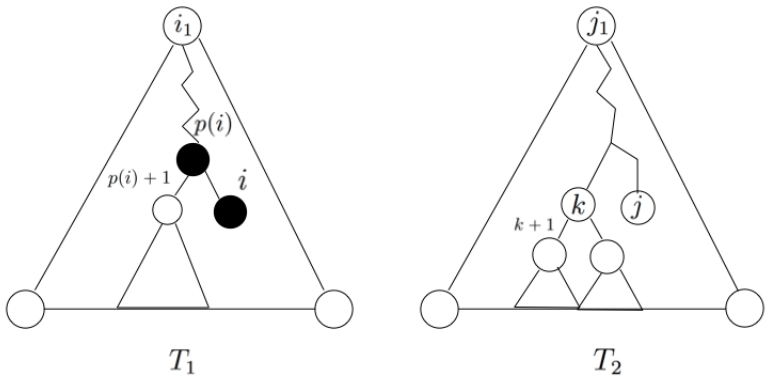

Figure 7: $p(i)$ is a gap node and i is its right child. Gap nodes are labeled black.

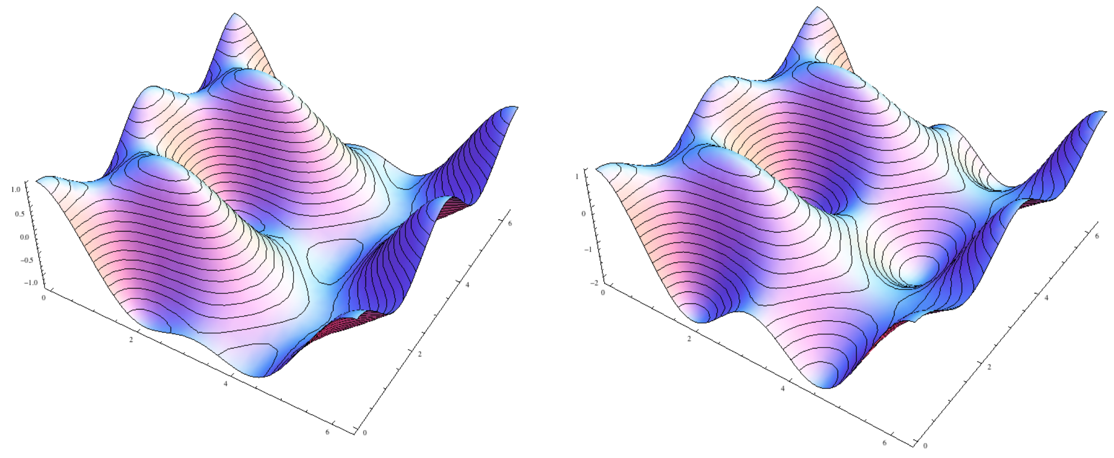

Figure 8: Two terrains.

• Here is the algorithm (point to the algorithm in the slides) with running time $O(m^3n^2 + m^2n^3)$. It is similar to the algorithm for the complete subtree model.

4 Applications and Future Works: 4 minutes

4.1 Application of Complete Subtree Gap Edit Distance in Contour Tree Comparison

Now we briefly discuss an application of complete subtree gap edit distance in contour tree comparison. Suppose we have two terrains that we want to determine how similar they are (point to Figure 4.1 in the slides):

Recall that to compare two terrains, we can compare their contour trees. Here is a terrain with its contour tree (point to Figure 9):

It can be seen that noise in the input correspond to complete subtrees in the contour tree. Thus complete subtree gap edit distance is applicable. We can

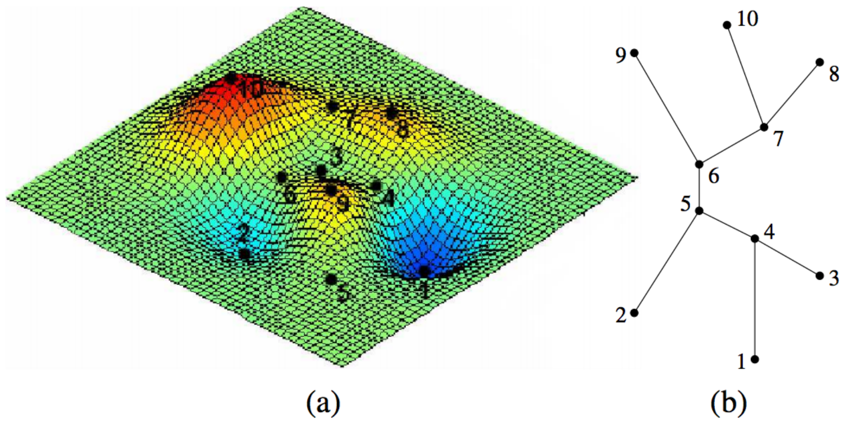

Figure 9: Terrain with its contour tree. Picture taken from P. Agarwal et al: *I/O-Effcient Batched Union-Find and Its Applications to Terrain Analysis*

use many geometric gap penalty functions, e.g. the height or the volume of the part of the terrain (due to noise) that correspond to a gap. We can also use topological penalty functions, e.g. the persistence of the part of the terrain that correspond to a gap.

Complete understanding of this application is still work in progress.

4.2 Future Works

- The recurrences for the general gap tree edit distance computations suggest that the reason for the arbitrary tree case be NP-hard is that there are too many branching factors, or degrees for each node. A natural next step is to study such distance between trees with fixed degree, e.g. ternary trees.

- In our model, gaps do not have any size constraints. For practical applications, it is more natural to have some upper bound on the gap size, as one usually has some control over the size of the noise. The question thus is: Can we incorporate that in the edit distance?

- The edit distance is a **topological similarity measure**. Thus the question is: It is possible to find a suitable geometric similarity measure, and combine it with the edit distance as in the trajectory comparison case?

References

[1] Pankaj K. Agarwal, Lars Arge, and Ke Yi. I/o efficient batched union-find and its applications to terrain analysis. *SCG '06: Proceedings of the 22nd Annual Symposium on Computational Geometry*, pages 167–176, 2006.

[2] Pankaj K. Agarwal, Herbert Edelsbrunner, John Harer, and Yusu Wang. Extreme elevation on a 2-manifold. *SCG '04: Proceedings of the ACM Symp. on Computational Geometry*, pages 357–365, 2004.

[3] Helmut Alt and M. Godau. Computing the frechet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5(12), 1995.

[4] Rolf Backofen, Shihyen Chen, Danny Hermelin, Gad M. Landau, Mikhail A. Roytberg, Oren Weimann, and Kaizhong Zhang. Locality and gaps in rna comparison. *Journal of Computational Biology*, 14(8):1074–1087, November 2007.

[5] Philip Bille. Asurvey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(2005):217–239, December 2004.

[6] G. Blin and H. Touzet. How to compare arc-annotated sequences: The alignment hierarchy. *SPIRE*, pages 291–303, 2006.

[7] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. *Comput. Geom.*, 24(2):75–94, 2003.

[8] Weimin Chen. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms*, 40:135–158, 2001.

[9] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications, 3rd edition.* Springer Verlag, Berlin, 2008.

[10] Erik D. Demaine, S. Mozes, B. Rossman, and O. Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Transactions on Algorithms*, 6(1), December 2009.

[11] Anne Driemel, S. Har-Peled, and Carola Wenk. Approximating the frechet distance for realistic curves in near linear time. *Proc. 26th Annu. ACM Sympos. Comput. Geom.*, pages 365–374, 2010.

[12] S. Dulucq and H. Touzet. Analysis of tree edit distance algorithms. *Proceedings of the 14th Annual Symposium Combinatorial Pattern Matching (CPM)*, pages 83–95, 2003.

[13] S. Dulucq and H. Touzet. Decomposition algorithms for the tree edit distance problem. *Journal of Discrete Algorithms*, 3:448–471, 2005.

[14] Herbert Edelsbrunner and John L. Harer. *Computational Topology.* American Mathematical Society, December 2009.

[15] T. Jiang, L. Wang, and K. Zhang. Alignment of trees - an alternative to tree edit. *Theor. Comput. Sci.*, 143(1):137–148, 1995.

[16] P. Klein. Computing the edit distance between unrooted ordered trees. *Proceedings of 6th European Symposium on Algorithms*, pages 91–102, 1998.

[17] S. Sankararaman, P. Agarwal, T. Mølhave, J. Pan, and A. P. Boedihardjo. Model-driven matching and segmentation of trajectories. *Proc. Twenty-Second ACM Symp. Advances Geographic Information Systems (to appear)*, 2013.

[18] S. Sankararaman, Pankaj. Agarwal, and T. Molhave. Computing similarity between a pair of trajectories (preprint). *http://arxiv.org/abs/1303.1585*, 2013.

[19] S. Schirmer and R. Giegerich. Forest alignment with affine gaps and anchors. *CPM 2011, LNCS*, 6661(104-117), 2011.

[20] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology.* PWS Publishing Company, Boston, 1997.

[21] B. Shapiro and K. Zhang. Comparing multiple rna secondary structures using tree comparisons. *Comput. Appl. Biosciences*, 4(3):387–393, 1988.

[22] H. Touzet. Tree edit distance with gaps. *Information Processing Letters*, 85(3):123–129, 2003.

[23] H. Xu. Point sets, curves and surfaces: A survey on shape matching and classification. Computational Geometry Course Project, December 2011.

[24] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIMA J. Comput.*, 18(6):1245–1262, 1989.

[25] Kaizhong Zhang and Dennis Shasha. *Tree Pattern Matching*, chapter 11, pages 341–371. Oxford University Press, 1997.

[26] Kaizhong Zhang, R. Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42:133–139, 1992.