Information Systems 56 (2016) 157–173

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Tree edit distance: Robust and memory-efficient

Mateusz Pawlik*, Nikolaus Augsten

University of Salzburg, Department of Computer Sciences, Jakob-Haringer-Str. 2, 5020 Salzburg, Austria

ARTICLE INFO

Article history:
Received 12 August 2014
Received in revised form
25 May 2015
Accepted 2 August 2015
Available online 28 August 2015

Keywords:
Tree edit distance
Similarity search
Approximate matching

ABSTRACT

Hierarchical data are often modelled as trees. An interesting query identifies pairs of similar trees. The standard approach to tree similarity is the tree edit distance, which has successfully been applied in a wide range of applications. In terms of runtime, the state-of-the-art algorithm for the tree edit distance is RTED, which is guaranteed to be fast independent of the tree shape. Unfortunately, this algorithm requires up to twice the memory of its competitors. The memory is quadratic in the tree size and is a bottleneck for the tree edit distance computation.

In this paper we present a new, memory efficient algorithm for the tree edit distance, AP-TED (All Path Tree Edit Distance). Our algorithm runs at least as fast as RTED without trading in memory efficiency. This is achieved by releasing memory early during the first step of the algorithm, which computes a decomposition strategy for the actual distance computation. We show the correctness of our approach and prove an upper bound for the memory usage. The strategy computed by AP-TED is optimal in the class of all-path strategies, which subsumes the class of LRH strategies used in RTED. We further present the AP-TED$^+$ algorithm, which requires less computational effort for very small subtrees and improves the runtime of the distance computation. Our experimental evaluation confirms the low memory requirements and the runtime efficiency of our approach.

1. Introduction

Data with hierarchical dependencies are often modelled as trees. Tree data appear in many applications, ranging from hierarchical data formats like JSON or XML to merger trees in astrophysics [33]. An interesting query computes the similarity between two trees. The standard measure for tree similarity is the tree edit distance, which is defined as the minimum-cost sequence of node edit operations that transform one tree into another. The tree edit distance has been successfully applied in bioinformatics (e.g., to find similarities between RNA secondary structures [1,29], neuronal cells [21], or glycan structures [3]), in image analysis [7], pattern recognition [25], melody recognition [19], natural language processing [28], information extraction [12,23], and document retrieval [22], and has received considerable attention from the database community [5,8–11,16–18,26,27].

* Corresponding author. Tel.: +43 66280446348.
E-mail addresses: mateusz.pawlik@sbg.ac.at (M. Pawlik), nikolaus.augsten@sbg.ac.at (N. Augsten).

http://dx.doi.org/10.1016/j.is.2015.08.004
0306-4379/

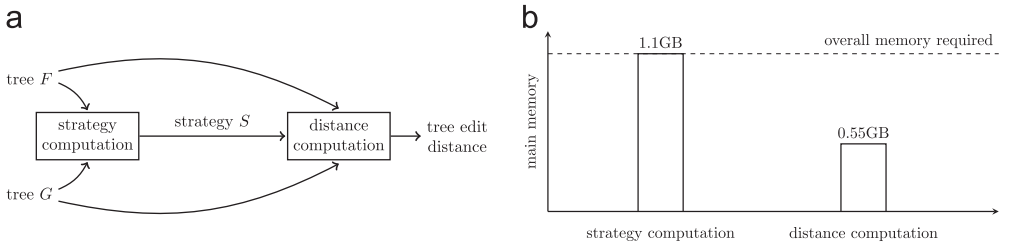

Fig. 1. Strategy computation requires more memory than actual distance computation. (a) Two-step algorithm for tree edit distance and (b) strategy vs. distance computation.

The fastest algorithms for the tree edit distance (TED) decompose the input trees into smaller subtrees and use dynamic programming to build the overall solution from the subtree solutions. The key difference between various TED algorithms is the decomposition strategy, which has a major impact on the runtime. Early attempts to compute TED [13,24,37] use a hard-coded strategy, which disregards or only partially considers the shape of the input trees. This may lead to very poor strategies and asymptotic runtime differences of up to a polynomial degree. The most recent development is the Robust Tree Edit Distance (RTED) algorithm [30], which operates in two steps (cf. Fig. 1(a)). In the first step, a decomposition strategy is computed. The strategy adapts to the input trees and is shown to be optimal among all previously proposed strategies. The actual distance computation is done in the second step, which executes the strategy.

In terms of runtime, the overhead for the strategy computation in RTED is small compared to the gain due to the better strategy. Unfortunately, this does not hold for the main memory consumption. Fig. 1(b) shows the memory usage for two example trees (perfect binary trees) of 8191 nodes: the strategy computation requires 1.1 GB of RAM, while the execution of the strategy (i.e., the actual distance computation) requires only 0.55 GB. Thus, for large instances, the strategy computation is the bottleneck and the fallback is a hard-coded strategy. This is undesirable since the gain of a good strategy grows with the instance size. Reducing the memory requirements of the strategy computation affects the maximum tree size that can be processed. This is crucial especially for large trees like abstract syntax trees of source code repositories [15,20] (Emacs: $>$10k nodes and MythTV: $>$50k nodes) or merger trees in astrophysics[1] [33].

In this paper we propose the AP-TED algorithm, which solves the memory problem of the strategy computation. This is achieved by computing the strategy bottom-up using dynamic programming and releasing part of the memorization tables early. We prove that our algorithm requires at most 1/3 of the memory that is needed by RTED's strategy computation [30]. As a result, the memory cost of the strategy computation is never above the cost of the distance computation. Our extensive experimental evaluation on various tree shapes, which require very different strategies, confirms our analytic memory bound and shows that our algorithm is often much better than its theoretical upper bound. For some tree shapes, it even runs in linear space, while the RTED strategy algorithm always requires quadratic space.

In addition to reducing the memory usage, AP-TED computes the optimum in a larger class of strategies than RTED. Strategies are expressed by root-leaf paths that guide the decomposition of the input trees. A path decomposes a tree into subtrees by deleting nodes and edges on a root-leaf path. Each resulting subtree is recursively decomposed by a new root-leaf path. RTED computes the optimal LRH strategy. An LRH strategy considers only left, right, and heavy paths. The left (right) root-leaf path connects each parent with its first (last) child; the heavy path connects the parent with the rightmost child that roots the largest subtree. AP-TED considers *all* root-leaf paths and is not limited to left, right, and heavy paths. Thus, our strategy is at least as good as the strategies used by RTED. To the best of our knowledge, this is the first algorithm to compute the optimal all-path strategy. The runtime complexity of our strategy algorithm is $O(n^2)$ as for the RTED strategy. This result is surprising since in each recursive step we need to consider a linear number of paths compared to only three paths (left, right, and heavy) in the RTED strategy. Our empirical evaluation suggests that in practice our strategy algorithm is even slightly faster than the RTED strategy algorithm since it allocates less memory.

On the distance computation side, we observe that a large number of subproblems that result from the tree decompositions are very small trees with one or two nodes only. We show that a significant boost can be achieved by treating these cases separately. We introduce the AP-TED$^+$ algorithm, which leverages that fact and achieves runtime improvements of more than 50% in some cases.

Summarizing, the contributions of this paper are the following:

- *Memory efficiency*. We substantially reduce the memory requirements w.r.t. previous strategy computation algorithms by traversing the trees bottom-up and systematically releasing memory early. The resulting AP-TED algorithm always consumes less memory for the strategy computation than for the actual distance computation and thus breaks the

[1] Accessible at http://www.mpa-garching.mpg.de/millennium/.