

# Fast alignment of fragmentation trees

Franziska Hufsky<sup>1,2</sup>, Kai Dührkop<sup>1</sup>, Florian Rasche<sup>1</sup>, Markus Chimani<sup>3</sup> and Sebastian Böcker<sup>1,\*</sup>

<sup>1</sup>Chair for Bioinformatics, Friedrich-Schiller-University, <sup>2</sup>Max Planck Institute for Chemical Ecology, Beutenberg Campus and <sup>3</sup>Algorithm Engineering, Friedrich-Schiller-University, Jena, Germany

## ABSTRACT

**Motivation:** Mass spectrometry allows sensitive, automated and high-throughput analysis of small molecules such as metabolites. One major bottleneck in metabolomics is the identification of ‘unknown’ small molecules not in any database. Recently, fragmentation tree alignments have been introduced for the automated comparison of the fragmentation patterns of small molecules. Fragmentation pattern similarities are strongly correlated with the chemical similarity of the molecules, and allow us to cluster compounds based solely on their fragmentation patterns.

**Results:** Aligning fragmentation trees is computationally hard. Nevertheless, we present three exact algorithms for the problem: a dynamic programming (DP) algorithm, a sparse variant of the DP, and an Integer Linear Program (ILP). Evaluation of our methods on three different datasets showed that thousands of alignments can be computed in a matter of minutes using DP, even for ‘challenging’ instances. Running times of the sparse DP were an order of magnitude better than for the classical DP. The ILP was clearly outperformed by both DP approaches. We also found that for both DP algorithms, computing the 1% slowest alignments required as much time as computing the 99% fastest.

**Contact:** sebastian.boecker@uni-jena.de

## 1 INTRODUCTION

Metabolomics deals with the identification and quantification of small compounds below 1000 Da, and has received increasing interest during the last years (Last *et al.*, 2007). All organisms, especially plants, fungi and bacteria, synthesize many different metabolites and a large portion of them is still unknown (Fernie *et al.*, 2004). The genome sequence usually does not reveal information about metabolite structure, as it does for protein structure. Newly identified metabolites often serve as leads in drug design (Li and Vederas, 2009; Schmidt *et al.*, 2007), in particular for antibiotics.

For a high-throughput analysis of metabolites and other small molecules, mass spectrometry (MS) is the predominant technology (Cui *et al.*, 2008; Fernie *et al.*, 2004; Last *et al.*, 2007). Nuclear magnetic resonance reveals more information about the analyte, but is inapplicable for high-throughput analysis due to its low sensitivity. MS can be coupled with a separation method (gas chromatography, liquid chromatography or capillary electrophoresis) to analyze complex mixtures like cell extracts (Fiehn, 2008; Halket *et al.*, 2005). To obtain information beyond the compound mass, the analyte is usually fragmented, and fragment masses are recorded. Typically, collision-induced dissociation is combined with liquid chromatography–MS, whereas gas chromatography–MS uses electron impact fragmentation. The first attempt to develop computational methods for analyzing fragmentation spectra of

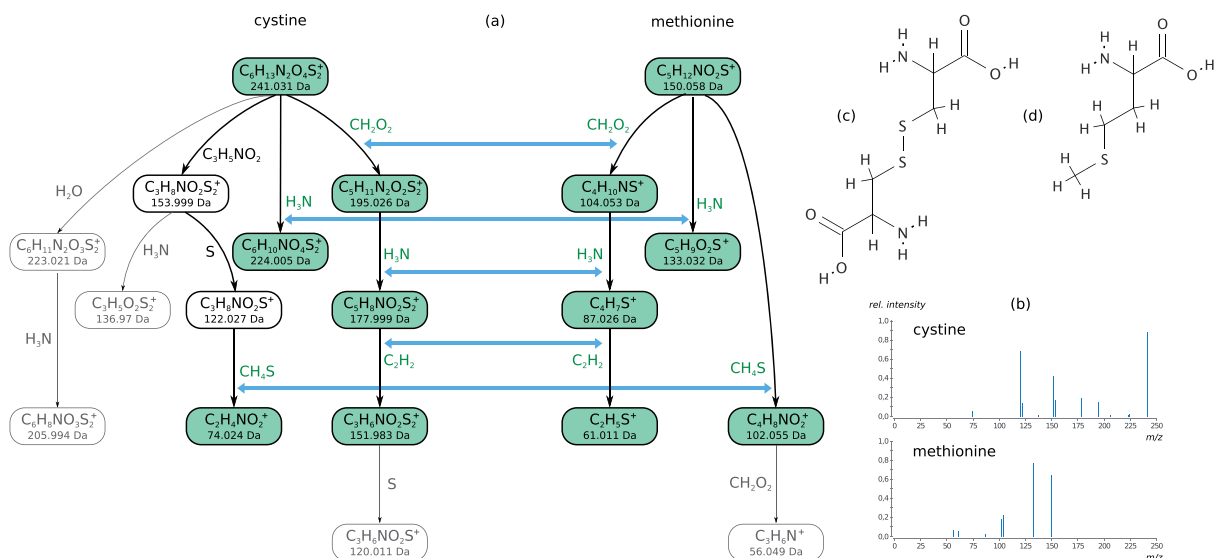
metabolites has been the DENDRAL project in 1965 (Lederberg, 1965). But the project was stopped after it became clear that automated structure elucidation using MS data could not be achieved at that time. Today, data analysis is still the major bottleneck in metabolomics (Neumann and Böcker, 2010). Manual data analysis requires time and deep knowledge of the underlying chemistry (Werner *et al.*, 2008). Due to the limited reproducibility of the data, even library searches are difficult (Oberacher *et al.*, 2009). Additionally, spectral libraries are vastly incomplete. Methods for *de novo* sequencing of non-ribosomal peptides have recently been developed. But these methods rely on the fact that the analytes are structurally restricted polymers with predictable fragmentation.

When manually analyzing tandem MS spectra, chemists try to annotate fragmentation peaks and identify relations between fragments, resulting in fragmentation pathways. This approach has been automated by Böcker and Rasche (2008). They calculate hypothetical fragmentation trees solely based on the MS data. Fragmentation tree nodes are annotated with the molecular formula of the fragments, whereas edges represent losses. To find a fragmentation tree that shows maximum agreement with the measured data, we search for a tree of maximum edge weight, comparable to prize-collecting Steiner trees (Ljubić *et al.*, 2005). Only lists of common and implausible losses are required as expert knowledge about fragmentation mechanisms. In Rasche *et al.* (2011), experts evaluated the calculated fragmentation trees and confirmed their excellent quality. Recently, methods to calculate fragmentation trees from multiple MS and gas chromatography–MS data have been developed (Hufsky *et al.*, 2012, manuscript; Scheubert *et al.* 2011).

MS analysis of similar compounds results in similar fragmentation trees. Rasche *et al.* (2012) proposed local tree alignments for the automated comparison of fragmentation trees and showed that this method is superior to spectral comparison. Fragmentation tree alignments even allow for inter-dataset comparisons for datasets measured on different instruments (Rasche *et al.*, 2012). A tree alignment may contain matches, mismatches, insertions and deletions, but respects the structure of the two trees. Fragmentation tree similarity is defined via edges (representing losses) and nodes (representing fragments). A local tree alignment contains those parts of the two trees where similar fragmentation cascades occurred.

Tree alignments were introduced by Jiang *et al.* (1995) and can be applied for RNA secondary structure comparison (Le *et al.*, 1989). RNA structure trees are ordered, that is, the children of any node have a fixed order. In contrast, fragmentation trees are unordered, as there cannot exist any sensible ordering for the sub fragments of some fragment. In this respect, fragmentation trees are more similar to phylogenetic trees than to RNA structure trees. Whereas efficient, polynomial-time algorithms exist for the alignment of ordered trees, the alignment of unordered trees is computationally hard, namely MAX SNP-hard (Jiang *et al.*, 1995). This implies that there exists no

\*To whom correspondence should be addressed.



**Fig. 1.** Optimal fragmentation tree alignment for cystine (11 losses) and methionine (6 losses) from the *Orbitrap* dataset (a). (b) Fragmentation mass spectra of cystine and methionine. The mass spectra do not share peaks. Molecular structures of cystine (c) and methionine (d). The molecular structures are not known to the alignment method. The alignment detects the common fragmentation path of formic acid–ammonia–ethylene losses and the separate ammonia branch. Additionally, it finds the methylthiol loss, which occurs at a later stage in cystine

Polynomial Time Approximation Scheme (PTAS) for the problem unless  $P=NP$  (Arora *et al.*, 1998). In case both trees have fixed maximum out degree, an optimum alignment can be computed via dynamic programming (DP) in polynomial time (Jiang *et al.*, 1995). In comparison, computing the edit distance between two unordered trees remains MAX SNP hard even for bounded degrees (Zhang and Jiang, 1994). An informal algorithm for aligning fragmentation trees was presented by Rasche *et al.* (2012), and JOIN nodes were introduced to account for missing nodes in one of the trees compared. Missing nodes result from missing peaks in one of the spectra. Rasche *et al.* (2012) do not give a correctness proof or running time analysis for the algorithm. As related work, we mention the TREE-CONSTRAINED BIPARTITE MATCHING problem where either a node or any of its descendants may be part of the matching; this problem is also APX-hard (Canzar *et al.*, 2011).

*Our contribution:* we modify the tree alignment algorithm from Jiang *et al.* (1995) for edge similarities and local alignments, and analyze its running time. We then show how to integrate JOIN nodes without increasing the worst-case running time. Next, we modify the algorithm to allow for sparse DP, a technique common in RNA folding (Backofen *et al.*, 2011). This does not only decrease the practical memory requirements of the method but, more importantly, also severely decreases running times in practice. Furthermore, we present an Integer Linear Program (ILP) for the fragmentation tree alignment problem, as ILPs often solve NP-hard problems swiftly in practice. We then evaluate all methods on real-world data, and find that the sparse DP approach dominates the classical DP, resulting in an 11-fold speed-up for one dataset. Somewhat unexpectedly, the ILP is clearly outperformed by both DP approaches.

## 2 ALIGNING FRAGMENTATION TREES

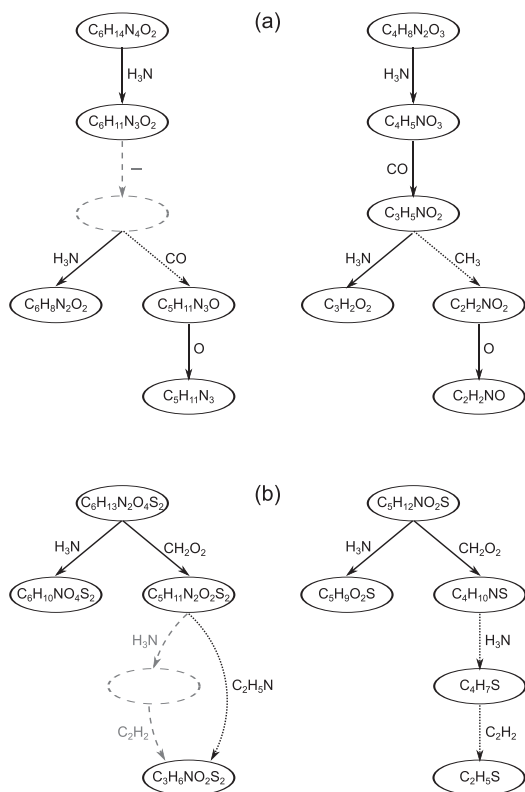
Let  $T=(V,E)$  be a tree; the elements of  $V$  are called *nodes*. In the following, we assume all trees to be *arboreal*, that is, all edges

in  $E$  are directed away from some root node. By  $uv$ , we denote a directed edge from node  $u$  to node  $v$ . Let  $p(v)$  be the parent node of some node  $v$ . A *fragmentation tree* is a tree  $T=(V,E)$  together with an edge labeling  $\ell:E\rightarrow\mathcal{L}$ . In more detail, nodes  $V$  correspond to fragments of a compound, whereas the labels  $\mathcal{L}$  are (neutral or radical) losses: that is, both are molecular formulas over some fixed alphabet of elements (Böcker and Rasche, 2008). Formally, molecular formulas are compomers or multi sets. The children of any node of a fragmentation tree are intrinsically unordered, as there is no sensible way to order the sub fragments of some fragment. In our presentation, we will not consider the molecular formulas of fragments, and concentrate on comparing losses only. As we will see below, comparing fragments instead or comparing both simultaneously requires only minor modifications. See Figure 1 for two examples of fragmentation trees.

We introduce some notation used throughout this article: Let  $T_1=(V_1,E_1)$  and  $T_2=(V_2,E_2)$  be the two trees we want to align. We sometimes call  $T_1$  the *left tree* and  $T_2$  the *right tree*. Let  $C(v)$  denote the children (nodes) of any node  $v$  in  $T_1$  or  $T_2$ . In the following, we usually assume that  $u$  is a node of  $T_1$ , and  $v$  a node of  $T_2$ . For  $i=1,2$ , let  $n_i:=|V_i|$  be the number of nodes in  $T_i$ , and let  $d_i$  be the maximum out degree in  $T_i$ . These maximum out degrees will be of particular interest to us, as the running time of our DP grows exponentially in  $d_1,d_2$ . Let  $\delta=\min\{d_1,d_2\}$  and  $\Delta=\max\{d_1,d_2\}$ .

Rasche *et al.* (2012) introduce a similarity function  $\sigma:\mathcal{L}\times\mathcal{L}\rightarrow\mathbb{R}$  for pairs of losses (molecular formulas). We do not repeat the details here, but note that this implies a similarity function  $\sigma:E_1\times E_2\rightarrow\mathbb{R}$  between edges of the two trees  $T_1,T_2$  via  $\sigma(e_1,e_2)=\sigma(\ell(e_1),\ell(e_2))$ . They also introduce a similarity function for fragment molecular formulas, which induces a similarity function between nodes of the two trees.

Furthermore, they also extend the definition of tree alignments by introducing a JOIN operator [see Figure 2(b)]: Given a path  $p_1$  in  $T_1$  of length two, let  $e_1,e'_1$  be the edges of  $p_1$ . We can assign a loss to



**Fig. 2.** Two alignments of fragmentation trees based on edge similarities. Nodes represent molecular formulas of the fragments, edges represent molecular formulas of the losses. **(a)** A gap (–) is introduced for the missing CO loss in the left tree (dashed edge and node). Losses CO and CH<sub>3</sub> are aligned by a mismatch (dotted edges). **(b)** In the left tree, the fragment after losing H<sub>3</sub>N is missing (dashed edges and node), whereas the fragment after further loss of C<sub>2</sub>H<sub>2</sub> is observed. To account for missing fragments, we introduce the join operation. It allows to align the two successive losses H<sub>3</sub>N and C<sub>2</sub>H<sub>2</sub> in the right tree to a single loss C<sub>2</sub>H<sub>5</sub>N in the left tree (dotted edges). Fragments may be missing because the corresponding peak was not detected, for example

$p_1$  by adding the corresponding losses  $\ell(e_1) + \ell(e'_1) \in \mathcal{L}$ . This means taking the sum of the respective compounds or the additive union of the corresponding multisets. We then assign a similarity between  $p_1$  and any edge  $e_2$  of  $T_2$  as  $\sigma(p_1, e_2) = \sigma(\ell(e_1) + \ell(e'_1), \ell(e_2))$ . Analogously, we can define a similarity for paths of length two in  $T_2$ . Obviously, this can be generalized to paths of arbitrary lengths but here, we will limit ourselves to paths of length two. For joining nodes in the alignment, we assume homogeneous join costs: The penalty for joining a node is  $\sigma_{\text{join}} \leq 0$ , independent of the node or edge that we want to join. Formally, this allows us to focus on the important aspects of our algorithms, and omit some technical details. Practically, we currently see no biologically reasonable way to assign different scores to different join nodes, as these usually correspond to the non-detection of a peak in one of the mass spectra.

Let  $T_1, T_2$  be two trees. We define a global alignment  $\mathcal{A}$  of  $T_1, T_2$  as follows (Jiang *et al.*, 1995):  $\mathcal{A}$  is a tree where nodes are labeled with pairs from  $(V_1 \cup \{-\}) \times (V_2 \cup \{-\})$ . Here, ‘–’ is the gap symbol [see Figure 2(a)]. If we restrict labels of  $\mathcal{A}$  to the first coordinate and contract all edges that end in a node labeled ‘–’, we end up with the

tree  $T_1$ ; if we do the same for the second coordinate, we end up with the tree  $T_2$ . (In fact, we have to replace the nodes of the restricted trees by their labels, we omit the simple technical details.) We say that  $\mathcal{A}$  is a local alignment if the trees originating from contracting gap edges are induced subtrees of  $T_1$  and  $T_2$ , respectively.

Different from Jiang *et al.* (1995), we want to score an alignment based on the edges of the two trees. To this end, for any node  $a$  of  $\mathcal{A}$  but the root, let  $e_1(a)$  be the unique edge in  $T_1$  that ends in the first coordinate of the label of  $a$ , and let  $e_2(a)$  be the unique edge in  $T_2$  that ends in the second coordinate of the label of  $a$ . In case no such edge exists, we assume  $e_1(a) = \text{‘-’}$  or  $e_2(a) = \text{‘-’}$ , respectively. Now, we define the score of  $\mathcal{A}$  as

$$\sum_{\text{non-root node } a \text{ of } \mathcal{A}} \sigma(e_1(a), e_2(a)).$$

We define  $\sigma(T_1, T_2)$  as the maximum score of a local alignment of  $T_1$  and  $T_2$ .

Scoring node pairs and scoring edge pairs are closely related: We can push an edge score into its end node, or we can pull a node score into its unique incoming edge. The only difference is that the root node is not considered when scoring edge pairs. The two scorings can be combined by introducing a particular root scoring  $\sigma^* : V_1 \times V_2 \rightarrow \mathbb{R}$  for the root nodes of the alignment. In the following, we omit the simple but somewhat tedious details, and simply note that all algorithms presented here work both with node scoring, edge scoring, as well as a combination thereof. A local fragmentation tree alignment for two compounds from the *Orbitrap* dataset (see Section 6) can be found in Figure 1.

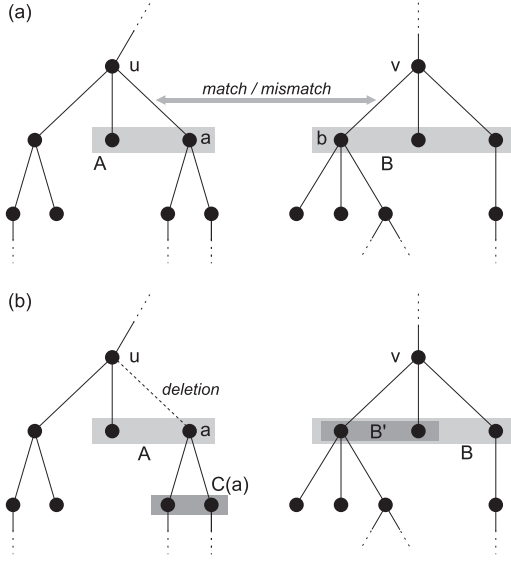
### 3 DYNAMIC PROGRAMMING

We now present an algorithm to compute optimum fragmentation tree alignments that has reasonable running time in practice. Our algorithm is a modification of an algorithm by Jiang *et al.* (1995) for computing global alignments of unordered trees. The reason for these algorithms to be swift in practice is that fragmentation trees usually have comparatively small out degree: fragments rarely have more than, say, five child fragments. We can limit the inevitable exponential part of the running time to this out degree.

We use DP to compute the maximal score  $\sigma(T_1, T_2)$  of a local alignment between two trees  $T_1, T_2$ . Let  $S(u, v)$  be the maximal score of a local alignment of two subtrees of  $T_1, T_2$ , where the subtree of  $T_1$  is rooted in  $u$ , and the subtree of  $T_2$  is rooted in  $v$ . For  $A \subseteq C(u)$  and  $B \subseteq C(v)$ , we define  $S_{u,v}[A, B]$  to be the score of an optimal local alignment of subtrees rooted in  $u$  and  $v$ , respectively, such that maximally the children  $A$  of  $u$  and  $B$  of  $v$  are used in the alignment. Clearly,  $S(u, v) = S_{u,v}[C(u), C(v)]$ . Furthermore, we have  $S_{u,v}[A, \emptyset] = S_{u,v}[\emptyset, B] = 0$  for all  $A, B$ . When all  $S(u, v)$  are known, we can compute the maximal score of a local alignment of  $T_1, T_2$  as

$$\sigma(T_1, T_2) = \max_{u \in T_1, v \in T_2} S(u, v). \quad (1)$$

We present a recurrence for the computation of  $S_{u,v}[A, B]$ . We initialize  $S_{u,v}[A, B] = 0$  for  $A = \emptyset$  or  $B = \emptyset$ . Recall that  $T_1$  is the left tree and  $T_2$  is the right tree. In the recurrence, we distinguish three cases, namely match (including mismatches), deletion left or deletion right, where the latter two are symmetric (Figure 3). For



**Fig. 3.** Representation of the match and the deleteL recurrences of the DP algorithm. (a)  $match_{u,v}[A, B]$  is the best score of matching edge  $ua$  on edge  $vb$ , such that maximally the children  $A$  of  $u$  and  $B$  of  $v$  are used. (b)  $deleteL_{u,v}[A, B]$  is the best score for deleting edge  $ua$ , such that maximally the children  $A$  of  $u$  and  $B$  of  $v$  are used. A subset  $B' \subseteq B$  of the children of  $v$  can now be matched to the children of  $a$

non-empty sets  $A \subseteq C(u)$  and  $B \subseteq C(v)$  we set

$$S_{u,v}[A, B] = \max \left\{ 0, match_{u,v}[A, B], deleteL_{u,v}[A, B], deleteR_{u,v}[A, B] \right\}$$

where we define

$$\begin{aligned} match_{u,v}[A, B] &:= \max_{a \in A, b \in B} \left\{ S(a, b) + S_{u,v}[A - \{a\}, B - \{b\}] + \sigma(ua, vb) \right\} \\ deleteL_{u,v}[A, B] &:= \max_{a \in A, B' \subseteq B} \left\{ S_{u,v}[C(a), B'] + S_{u,v}[A - \{a\}, B - B'] + \sigma(ua, -) \right\} \\ deleteR_{u,v}[A, B] &:= \max_{A' \subseteq A, b \in B} \left\{ S_{u,v}[A', C(b)] + S_{u,v}[A - A', B - \{b\}] + \sigma(-, vb) \right\} \end{aligned} \quad (2)$$

Here,  $\sigma(ua, vb)$  denotes the score of the losses attached to arcs  $ua$  and  $vb$ , and  $\sigma(ua, -), \sigma(-, vb)$  accordingly. Recurrence (2) is the obvious modification of the recurrence presented in Jiang *et al.* (1995) for global alignments and node similarities.

Merging two losses in  $T_1$  or  $T_2$  requires two additional symmetric cases, namely join left and join right for merging in tree  $T_1$  or  $T_2$ , respectively. To speed up computations, we add an additional

PREJOIN case for nodes that will be joined in the alignment. We set

$$S_{u,v}[A, B] = \max \left\{ 0, match_{u,v}[A, B], deleteL_{u,v}[A, B], deleteR_{u,v}[A, B], joinL_{u,v}[A, B], joinR_{u,v}[A, B] \right\} \quad (3)$$

where we define, in addition to (2),

$$\begin{aligned} prejoinL_{u,v}[A, B] &:= \max_{a \in A, b \in B} \left\{ S(a, b) + prejoinL_{u,v}[A - \{a\}, B - \{b\}] + \sigma(p(u)a, vb) + \sigma_{join} \right\} \\ joinL_{u,v}[A, B] &:= \max_{a \in A, B' \subseteq B} \left\{ prejoinL_{a,v}[C(a), B'] + S_{u,v}[A - \{a\}, B - B'] \right\} \end{aligned} \quad (4)$$

Here,  $\sigma(p(u)a, vb)$  is the score for the combined losses on the path from  $p(u)$  to  $a$  with the loss of edge  $vb$ . Recall that  $\sigma_{join} \leq 0$  is the penalty for joining a node. Again, we initialize  $joinL_{u,v}[A, \emptyset] = joinL_{u,v}[\emptyset, B] = 0$  for all  $A, B$ . Analogously to (4), we can define recurrences for  $prejoinR_{u,v}[A, B]$  and  $joinR_{u,v}[A, B]$ .

For bottom-up DP (Sniedovich, 2006), we have to find an order in which the entries of the DP tables can be filled. Computation of  $match_{u,v}[A, B]$ ,  $deleteL_{u,v}[A, B]$  and  $deleteR_{u,v}[A, B]$  only accesses entries  $S_{u',v'}[A', B']$ , such that  $u' \in \{u\} \cup C(u)$  and  $v' \in \{v\} \cup C(v)$ . By processing nodes in postorder, we ensure that all  $S_{u',v'}[A', B']$  are previously computed for  $(u', v') \neq (u, v)$ . For the remaining case, we iterate  $|A| + |B| = 0, 1, \dots, |C(u)| + |C(v)|$ . Similar arguments hold for the computation of JOIN and PREJOIN nodes.

**THEOREM 1.** Let  $T_1 = (V_1, E_1)$  and  $T_2 = (V_2, E_2)$  be two trees,  $\sigma : E_1 \cup \{-\} \times E_2 \cup \{-\} \rightarrow \mathbb{R}$  a scoring function between edge pairs, and  $\sigma_{join} \in \mathbb{R}$  the penalty for joining a node. For  $i = 1, 2$  set  $n_i := |V_i|$ , and let  $d_i$  be the maximum out degree in  $T_i$ . The maximum score  $\sigma(T_1, T_2)$  of a local alignment of  $T_1, T_2$  can be computed in  $O(3^\Delta \cdot 2^\delta \cdot \delta n_1 n_2)$  using recurrence (3) and equation (1), where  $\Delta := \max\{d_1, d_2\}$  and  $\delta := \min\{d_1, d_2\}$ .

The proof of the theorem is based on the following lemma:

**LEMMA 1.** Computing  $S_{u,v}[A, B]$  for all  $A \subseteq C(u)$  and  $B \subseteq C(v)$  is possible using recurrence (3) in  $O(3^{d_u} \cdot 2^{d_v} \cdot d_v + 2^{d_u} \cdot 3^{d_v} \cdot d_u)$  time, where  $d_u = |C(u)|$  and  $d_v = |C(v)|$ .

See the Supplementary Material for proofs of Lemma 1 and Theorem 1. Similarly to Theorem 1, we can show that any pairwise tree alignment that does not take joining nodes into account, can also be computed in this time. We leave out the straightforward details.

**THEOREM 2.** A pairwise unordered tree alignment (global or local, scoring nodes or edges or both, with similarities or costs) of rooted trees  $T_1, T_2$  can be computed in  $O(3^\Delta \cdot 2^\delta \cdot \delta n_1 n_2)$  time. Here,  $n_i$  is the number of nodes in tree  $T_i$ , and  $d_i$  is the maximum out degree in  $T_i$ , for  $i = 1, 2$ ; furthermore,  $\Delta := \max\{d_1, d_2\}$  and  $\delta := \min\{d_1, d_2\}$ .

#### 4 SPARSE DYNAMIC PROGRAMMING

Applying the above algorithm to real-world instances of aligning fragmentation trees, one can see that  $S(u, v) = 0$  holds for many node pairs  $u, v$ . This can be attributed to two factors: First, we are computing local alignments, so we can always choose to end the alignment subtrees in the nodes  $u, v$ . Second, there are many different labels found at the edges (or nodes) of a fragmentation tree. A reasonable scoring scheme will assign negative scores to most non-matching edge (or node) labels, so it is rather the exception than the rule that we can find two nodes  $u, v$  with  $S(u, v) > 0$ .

The idea is to ‘sparsify’ our DP tables by storing only those table entries with positive values. Thereby, we face the following fact: If  $S_{u,v}[A, B] > 0$  for  $A \subseteq C(u)$  and  $B \subseteq C(v)$  then  $S_{u,v}[A', B'] > 0$  holds for all supersets  $A', B'$  with  $A \subseteq A' \subseteq C(u)$  and  $B \subseteq B' \subseteq C(v)$ . So, as soon as we have one non-zero entry in the table, then an exponentially large part of the table will be filled with non-zero entries, too.

To negate this rather unfortunate effect, we modify our DP as follows: for  $A \subseteq C(u)$  and  $B \subseteq C(v)$ , we define  $S'_{u,v}[A, B]$  to be the score of an optimum local alignment with subtrees rooted in  $u$  and  $v$ , respectively, such that exactly the children  $A$  of  $u$  and  $B$  of  $v$  are used in the local alignment. If no such alignment exists, we set  $S'_{u,v}[A, B] = -\infty$ . Then  $S'_{u,v}[\emptyset, \emptyset] = 0$ , but for all  $A, B \neq \emptyset$  we have  $S'_{u,v}[A, \emptyset] < 0$ ,  $S'_{u,v}[\emptyset, B] < 0$ . Clearly,

$$S(u, v) = \max_{A \subseteq C(u), B \subseteq C(v)} S'_{u,v}[A, B]. \quad (5)$$

We need one more trick in our recurrence: in (2) we have accessed entries  $S_{a,v}[C(a), B']$  and  $S_{u,b}[A', C(b)]$ , but this is not possible for the table  $S'$  as the optimal alignments might not use all the children of  $a$  or  $b$ . To this end, we introduce

$$S'_{u,v}[A, *] := \max_{B' \subseteq C(v)} \left\{ S'_{u,v}[A, B'] \right\},$$

$$S'_{u,v}[* , B] := \max_{A' \subseteq C(u)} \left\{ S'_{u,v}[A', B] \right\},$$

for the maximum over all subsets of  $C(v)$  or  $C(u)$ , respectively. For non-empty sets  $A \subseteq C(u)$  and  $B \subseteq C(v)$  we set

$$S'_{u,v}[A, B] = \max \left\{ \begin{aligned} &\text{match}'_{u,v}[A, B], \\ &\text{delete}L'_{u,v}[A, B], \text{delete}R'_{u,v}[A, B], \\ &\text{join}L'_{u,v}[A, B], \text{join}R'_{u,v}[A, B] \end{aligned} \right\} \quad (6)$$

which, compared to (3), misses the lower bound 0 and uses the definitions:

$$\begin{aligned} \text{match}'_{u,v}[A, B] &:= \max_{a \in A, b \in B} \left\{ S(a, b) + \right. \\ &\quad \left. S'_{u,v}[A - \{a\}, B - \{b\}] + \sigma(ua, vb) \right\} \\ \text{delete}L'_{u,v}[A, B] &:= \max_{a \in A, B' \subseteq B} \left\{ S'_{a,v}[* , B'] + \right. \\ &\quad \left. S'_{u,v}[A - \{a\}, B - B'] + \sigma(ua, -) \right\} \\ \text{delete}R'_{u,v}[A, B] &:= \max_{A' \subseteq A, b \in B} \left\{ S'_{u,b}[A', *] + \right. \\ &\quad \left. S'_{u,v}[A - A', B - \{b\}] + \sigma(-, vb) \right\} \end{aligned} \quad (7)$$

For the further join recurrences, we only concentrate on the left tree. The definition of  $\text{prejoin}L'_{u,v}[A, *]$  and the JOIN recurrences at the right tree are analogous.

$$\begin{aligned} \text{prejoin}L'_{u,v}[A, B] &:= \max_{a \in A, b \in B} \left\{ S(a, b) + \right. \\ &\quad \left. \text{prejoin}L'_{u,v}[A - \{a\}, B - \{b\}] + \right. \\ &\quad \left. \sigma(p(u)a, vb) + \sigma_{\text{join}} \right\} \\ \text{prejoin}L'_{u,v}[* , B] &:= \max_{A' \subseteq A} \left\{ \text{prejoin}L'_{u,v}[A', B] \right\} \\ \text{join}L'_{u,v}[A, B] &:= \max_{a \in A, B' \subseteq B} \left\{ \text{prejoin}L'_{a,v}[* , B'] + \right. \\ &\quad \left. S'_{u,v}[A - \{a\}, B - B'] \right\} \end{aligned} \quad (8)$$

To summarize, the central point is that we do not have to store any entries with  $S'_{u,v}[A, B] \leq 0$ : such entries will never lead to an optimal alignment, as we are better off removing all nodes  $A, B$ , plus everything below these nodes from the alignment. The only exception to this rule is that we store the entry  $S'_{u,v}[\emptyset, \emptyset] = 0$ . Furthermore, we do not have to store entries  $S'_{u,v}[A, B]$  if there exist subsets  $A' \subseteq A$ ,  $B' \subseteq B$  with  $(A', B') \neq (A, B)$  such that  $S'_{u,v}[A, B] \leq S'_{u,v}[A', B']$ . In this case, we can replace an alignment that uses children  $A, B$  of  $u, v$ , by an alignment that uses only children  $A', B'$  and has better or equal score. We say that an entry  $S'_{u,v}[A, B]$  is dominated by entry  $S'_{u,v}[A', B']$ . For a scoring scheme that assigns negative scores for non-matching edge (or node) labels, large parts of the tables have negative scores or are dominated by another entry. We do not actually have to forbid that dominated entries are stored, as they do not interfere with our computations; rather, we are free to leave out dominated entries when we encounter them.

The resulting tables  $S'_{u,v}$  are sparsely populated, and for many vertices  $u, v$ , there are no entries with  $S'_{u,v}[A, B] > 0$ . We can reduce the memory consumption of the method using hash maps instead of arrays. Hash map implementations like Cuckoo hashing (Pagh and Rodler, 2004) or Hopscotch hashing (Herlihy *et al.*, 2008) can carry out all operations in constant (amortized) time. In practice, we find that memory consumption is usually not prohibitive. In this case, we can use lazy arrays that are not allocated until a first entry is stored.

*Resolving the recurrences:* Now, it is time for our final trick: instead of computing the scores using recurrence (6–8), we apply a successive approximation procedure similar to Dijkstra’s Algorithm for shortest paths (Sniedovich, 2006). That is, instead of ‘pulling’ scores from previously calculated entries, we ‘push’ scores from entries that have been finalized. For example, assume that we have finalized the computation of some entry  $S'_{u,v}[A, B]$  for fixed  $A \subseteq C(u)$  and  $B \subseteq C(v)$ . Also assume that  $S'_{u,v}[A, B] > 0$  as otherwise,  $S'_{u,v}[A, B]$  is dominated by  $S'_{u,v}[\emptyset, \emptyset] = 0$ . Then, recurrence (7) tells us that we can update other entries of the table accordingly: if  $S'_{u,v}[A, B] > S'_{u,v}[* , B]$  (which we assume to be incompletely calculated so far) then  $S'_{u,v}[* , B] \leftarrow S'_{u,v}[A, B]$ . Similarly, if  $S'_{u,v}[A, B] > S'_{u,v}[A, *]$  then  $S'_{u,v}[A, *] \leftarrow S'_{u,v}[A, B]$ , and if  $S'_{u,v}[A, B] > S(u, v)$  then  $S(u, v) \leftarrow S'_{u,v}[A, B]$ . Regarding the recurrence for  $\text{match}'$ , we iterate over all  $a \in C(u) \setminus A$  and  $b \in C(v) \setminus B$ : If  $\text{match}'_{u,v}[A \cup \{a\}, B \cup \{b\}] < S(a, b) + S'_{u,v}[A, B] + \sigma(ua, vb)$  then update it accordingly. If  $\text{match}'_{u,v}[A \cup \{a\}, B \cup \{b\}] \leq$

match'\_{u,v}[A, B] then the entry match'\_{u,v}[A \cup \{a\}, B \cup \{b\}] is dominated and we can remove it from the hash map.

For all other cases, similar updates can be performed, which we only sketch here: For deleteL' we iterate over all  $a \in C(u) \setminus A$  and  $B' \subseteq C(v) \setminus B$ ; if  $\text{deleteL}'_{u,v}[A \cup \{a\}, B \cup B'] < S'_{a,v}[*] + S'_{u,v}[A, B] + \sigma(u, a, -)$  then update it accordingly. Updates have to be performed as soon as an entry is finalized, that is, it cannot be changed by any future modifications. Finding finalized entries is similar to the order of computations in the previous section; we omit the technical details.

The above algorithm has exactly the same worst-case running time complexity as the initial recurrence from Section 3. But in practice, we can get even faster, at least in cases where the arrays are very sparse: to this end, finalizing some entry  $\text{deleteL}'_{u,v}[A, B]$  triggers updates for all subsets  $B' \subseteq C(v) \setminus B$ . But only those  $B'$  can lead to relevant updates where  $S'_{a,v}[*] + S'_{u,v}[A, B'] > 0$  holds. Otherwise, the updated entry will be dominated by  $S'_{a,v}[*] + S'_{u,v}[A, B] = 0$ . If we iterate over the hash map for those  $B'$  with  $S'_{a,v}[*] + S'_{u,v}[A, B'] > 0$  then the worst-case running time increases to  $O(4^\Delta \cdot 2^\delta \cdot \delta \cdot n_1 n_2)$ , assuming constant time access to the hash map. However, in practice, running time decreases if the DP tables are sparsely populated. We stress that the sparse DP still guarantees to find the optimal solution.

## 5 INTEGER LINEAR PROGRAMMING

ILPs are a classical approach for finding exact solutions of computationally hard problems. We now present an ILP for computing a pairwise unordered tree alignment. Again, let  $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$  be the input trees with  $V_1 \cap V_2 = \emptyset$ . As the ILP is edge based, we have to introduce some additional notation: Let  $e \in E_i, i \in \{1, 2\}$ , be any edge in one of the two given trees. We denote by  $\mathcal{D}(e)$  the set of edges in the subtree rooted at the head of  $e$ , and by  $\mathcal{N}(e) := E_i \setminus (\{e\} \cup \mathcal{D}(e))$  the non-descendant edges of  $e$ . For an edge  $e$ , we define  $p(e)$  to be the parent edge, and  $p^*(e) := \{p(e), p(p(e)), \dots\}$  all of its ancestor edges. Finally,  $\mathcal{F}(e) := \mathcal{D}(p(e)) \cap \mathcal{N}(e)$  is the 'extended family' of  $e$ , that is, all descendants of  $e$ 's parent edge, except for  $e$  and its descendants.

We start with the ILP without considering the join operation (ILP 1) and use the following binary variables: Iff an edge  $e' \in (E_1 \cup E_2)$  appears in the aligned subtree, we have  $z_{e'} = 1$ ; iff this edge is aligned to a gap, we have  $y_{e'} = 1$ . Finally, iff an edge  $e \in E_1$  is aligned to an edge  $f \in E_2$ , we have  $x_{\{e,f\}} = 1$ . The constraints (10) ensure for each edge that we decide whether this edge is used in the alignment and if, how it is aligned. The inequalities (11) ensure that the subgraphs of  $T_1$  (and  $T_2$ ) are proper trees. Finally, (12) ensure that the obtained alignments are consistent: assume an alignment  $(e, f)$  then we cannot also align a descendant of  $e$  with a non-descendant of  $f$  and vice versa. The conditional term following the universal quantifier simply avoids redundancy.

Based thereon, we can construct an ILP allowing join operations (ILP 2). Therefore, we require additional binary variables  $x_{\{e,f\}}^{(i)}$  (with  $i \in \{1, 2\}, e \in E_i, f \in E_{3-i}$ ), which are 1 iff the joined edges  $(p(e), e)$  are aligned with  $f$ . Technically, we also require  $x_{\{e,f\}} = 1$  in such a case. Note that this amount of additional variables is necessary to compose a linear objective function, when the join costs cannot be computed only based on align- and gap costs. Furthermore, we introduce binary variables  $\phi_{e'}, e' \in (E_1 \cup E_2)$ , which are 1 iff the edge  $e'$  is used as a parent edge within

**Table 1.** The three datasets used in this study

Characteristics of the datasets	Orbitrap	MassBank	Hill
Number of compounds	97	370	102
Number of non-empty trees	93	343	102
Maximum out degree	7	6	10
Average/median out degree <sub>max</sub>	3	2	5
Number of alignments	4278	58 653	5151

Fragmentation trees were computed for all compounds. Only non-empty trees were considered for tree alignment. The maximum out degree of a single tree is denoted by out degree<sub>max</sub>. Number of alignments is given without self-alignments.

a join (e.g.,  $\phi_{p(e)} = 1$  if the former  $x_{\{e,f\}}^{(i)}$  variable is 1). We use the shorthands  $\sigma^{(1)}(e, f) := \sigma(e + p(e), f) + \sigma_{\text{join}} - \sigma(e, f)$  and  $\sigma^{(2)}(e, f) := \sigma(e, f + p(f)) + \sigma_{\text{join}} - \sigma(e, f)$  in the objective function.

Constraints (15)–(17) are analogous to the former ILP. While (18) guarantees that joins are always separated from each other within an input tree, (19) ensures that at most one joined alignment may occur for any edge. Inequalities (20)–(22) make sure that a parent edge  $e'$  is only marked as a joined parent iff all its aligned children are joined with  $e'$ . Finally, (23) guarantees that we do not align two joined edges with each other.

## 6 EXPERIMENTAL RESULTS

To evaluate our work, we used three different test datasets (Table 1). The *Orbitrap* dataset (Rasche et al., 2012) contains 97 compounds, measured on a Thermo Scientific Orbitrap XL instrument. The *MassBank* dataset (Horai et al., 2010) consists of 370 compounds measured on a Waters Q-ToF Premier spectrometer. The *Hill* dataset consists of 102 compounds measured on a Micromass Q-ToF, published by Hill et al. (2008). We omit the experimental details. Fragmentation trees were computed using ILP as described in Rauf et al. (2012). Self-alignments were excluded from the analysis.

For our evaluations, we use a scoring function very similar to the one from (Rasche et al., 2012), evaluating pairs of losses and pairs of fragments. For losses  $nl_1, nl_2$ , we distinguish between size-dependent positive match scores  $\sigma(nl, nl) := 5 + \text{number of non-hydrogen atoms}$  and size-dependent negative mismatch scores  $\sigma(nl_1, nl_2) := -5 \text{ number of different non-hydrogen atoms}$ . For fragments  $f_1, f_2$ , we use size-dependent positive match scores  $\sigma(f, f) := 5 + \text{number of non-hydrogen atoms}$  and size-independent negative mismatch scores  $\sigma(f_1, f_2) := -3$ . We allow insertion/deletions, as well as joining two subsequent losses, both without penalty. The idea behind this *ad hoc* scoring is to reward or penalize large losses stronger than small losses, whereas non-matching fragments are penalized independent of size. See Rasche et al. (2012) for details.

We implemented the DP algorithms in Java 1.6. For the sparse DP, we used lazy arrays to store the DP tables. We solved the ILP via branch and cut using CPLEX 12.1 in its default settings. Computation was done on two different but comparable computers, namely on a quad-core 2.2 GHz AMD Opteron processor with 5 GB of main memory for the DP algorithms, and on a quad-core Intel Xeon E5520 with 2.27 GHz in 32-bit mode for the ILP, using 2 GB RAM per job. For the DP algorithms, we repeated computations five times, reporting the minimum running time for each instance.



$$\max \sum_{\substack{e \in E_1, \\ f \in E_2}} \sigma(e, f) \cdot x_{\{e, f\}} + \sum_{e' \in E_1} \sigma(e', -) \cdot y_{e'} + \sum_{e' \in E_2} \sigma(-, e') \cdot y_{e'} \quad (9)$$

$$\text{s.t.} \quad y_e + \sum_{f \in E_{3-i}} x_{\{e, f\}} = z_e \quad \forall i \in \{1, 2\}, e \in E_i \quad (10)$$

$$z_{e'} + z_{e''} \leq 1 + z_e \quad \forall i \in \{1, 2\}, e \in E_i, e' \in \mathcal{D}(e), e'' \in \mathcal{F}(e) \quad (11)$$

$$x_{\{e, f\}} + x_{\{e', f'\}} \leq 1 \quad \forall i \in \{1, 2\}, e \in E_i, f \in E_{3-i}, \quad (12)$$

$$e' \in \mathcal{D}(e), f' \in \mathcal{N}(f), [\text{if } i=2: f' \notin p^*(f)]$$

$$x_{\{e, f\}}, y_{e'}, z_{e'} \in \{0, 1\} \quad \forall e \in E_1, f \in E_2, e' \in (E_1 \cup E_2) \quad (13)$$

**ILP 1:** The ILP for pairwise unordered tree alignment without join operations

$$\max \sum_{\substack{e \in E_1, \\ f \in E_2}} \left( \sigma(e, f) x_{\{e, f\}} + \sum_{i \in \{1, 2\}} \sigma^{(i)}(e, f) x_{\{e, f\}}^{(i)} \right) + \sum_{e' \in E_1} \sigma(e', -) y_{e'} + \sum_{e' \in E_2} \sigma(-, e') y_{e'} \quad (14)$$

$$\text{s.t.} \quad y_e + \phi_e + \sum_{f \in V_{3-i}} x_{\{e, f\}} = z_e \quad \forall i \in \{1, 2\}, e \in E_i \quad (15)$$

$$z_{e'} + z_{e''} \leq 1 + z_e \quad \forall i \in \{1, 2\}, e \in E_i, e' \in \mathcal{D}(e), e'' \in \mathcal{F}(e) \quad (16)$$

$$x_{\{e, f\}} + x_{\{e', f'\}} \leq 1 \quad \forall i \in \{1, 2\}, e \in E_i, f \in E_{3-i}, \quad (17)$$

$$e' \in \mathcal{D}(e), f' \in \mathcal{N}(f), [\text{if } i=2: f' \notin p^*(f)]$$

$$\phi_{e'} + \phi_{e''} \leq 1 \quad \forall e' \in (E_1 \cup E_2), e'' = p(e') \quad (18)$$

$$x_{\{e, f\}}^{(1)} + x_{\{e, f\}}^{(2)} \leq x_{\{e, f\}} \quad \forall e \in E_1, f \in E_2 \quad (19)$$

$$x_{\{e, f\}} - x_{\{e, f\}}^{(i)} \leq 1 - \phi_{e'} \quad \forall i \in \{1, 2\}, e \in E_i, e' = p(e), f \in E_{3-i} \quad (20)$$

$$y_e \leq 1 - \phi_{e'} \quad \forall i \in \{1, 2\}, e \in E_i, e' = p(e) \quad (21)$$

$$x_{\{e, f\}}^{(i)} \leq \phi_{e'} \quad \forall i \in \{1, 2\}, e \in E_i, e' = p(e), f \in V_{3-i} \quad (22)$$

$$x_{\{e, f\}}^{(i)} + \phi_{f'} \leq 1 \quad \forall i \in \{1, 2\}, e \in E_i, f \in E_{3-i}, f' = p(f) \quad (23)$$

$$x_{\{e, f\}}, y_{e'}, z_{e'}, x_{\{e, f\}}^{(i)}, \phi_{e'} \in \{0, 1\} \quad \forall i \in \{1, 2\}, e \in E_1, f \in E_2, e' \in (E_1 \cup E_2) \quad (24)$$

**ILP 2:** The ILP for pairwise unordered tree alignment including join operations

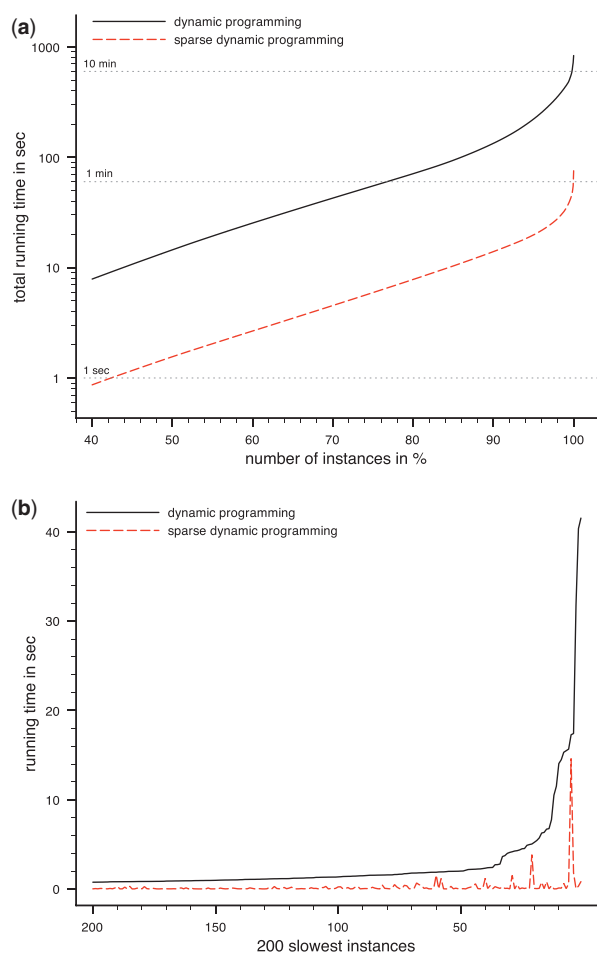
For the *Orbitrap* and the *MassBank* dataset, we found that for over 98% of the instances, the running time was in the range of microseconds for both DP algorithms. For these datasets, we only evaluate total running times for all alignments. For *MassBank*, the classical DP (Section 3) finished in 4.2s for an all-against-all alignment of 343 trees, whereas sparse DP (Section 4) only required 1.8s. For *Orbitrap*, the classical DP finished in 5.4s for the all-against-all alignment of 93 trees, whereas sparse DP required 0.6s, a 9-fold speed-up. In contrast, the ILP needed 9.6min for all alignments in the *MassBank* datasets and 14.5min for all alignments in the *Orbitrap* dataset.

The *Hill* dataset contains trees with much higher maximum out degree, so we performed a more detailed running time analysis. Classical DP required 13.9min and sparse DP finished in 1.3min, an 11-fold speed-up. Running times of the ILP could only be measured without allowing join operations. For 1241 instances, computations run into the memory limitation of 2GB. For the

remaining alignments, the ILP finished in 11.24h. Hence, we excluded the ILP from our detailed analysis. To get an overview of the differences in the running times between hard and easy alignments, we sorted the instances by their running times in increasing order. This was done separately for each algorithm. See Figure 4 (top) and Table 2. For both algorithms, we found that the 99% fastest alignments need nearly as much computing time as the remaining 1% slowest alignments. We further sorted all instances by the running time of the classical DP (see again Figure 4). We found that for every instance, sparse DP requires less time than the classical DP.

## 7 CONCLUSION

Fragmentation trees are a tool to overcome the limitations of spectral library search, as they, for the first time, enable us to retrieve not only exact hits, but also similar compounds from a spectral database. But performing the workflows proposed by Rasche *et al.* (2012) on



**Fig. 4.** Running times for the *Hill* dataset with 5151 individual alignments. (a) Total running times when instances are sorted by individual running times. For any fraction  $x\%$ , we calculate the total running time of the  $x\%$ , instances for which the alignment was computed faster than for any of the remaining instances. For example at 50% one can find the running time that was needed to compute the 50% fastest instances. For each algorithm, instances were sorted separately. Note the logarithmic y-axis. (b) Individual running times for the 200 slowest instances of the classical DP algorithm. Instances are sorted by their running time for the classical DP algorithm. One can see that running times of the classical DP are outperformed by that of the sparse DP

a large database requires tree alignments to be executed extremely fast. In this article, we have presented three exact algorithms for the alignment of fragmentation trees. We find that the sparse DP approach dominates the classical DP, resulting in an 11-fold speed-up for one dataset. ILPs have an excellent record of providing fast algorithms for NP-hard problems. Thus, it is rather unexpected that, for the problem discussed here, the ILP is usually clearly outperformed by both DP approaches; still, it has the potential to solve those instances that are ‘hard’ for DP-based algorithms. Also, in such cases we may use the ILP as a heuristic, solving only its LP relaxation and applying some integer rounding algorithm, many of which are standard in state-of-the-art ILP solvers.

**Table 2.** Running times for the *Hill* dataset

Algorithm	All	90% fastest	99% fastest	1% slowest
DP	833.3 s	133.5 s (16.0%)	437.9 s (52.6%)	395.4 s (47.4%)
Sparse DP	75.3 s	13.9 s (18.5%)	33.9 s (45.0%)	41.4 s (55.0%)
Speed up	11-fold	10-fold	13-fold	10-fold

We report running times in seconds and as fractions of the total running time for all instances (5151 alignments). We also report running time for the 90 and 99% fastest and for the 1% slowest alignments. For both algorithms, instances were sorted separately.

When larger datasets become available, we expect the total running time of an all-against-all alignment to increase more than quadratic with dataset size: We have shown above that a large fraction of the total running time stems from a few ‘hard’ alignments which, in turn, correspond to a few trees in the dataset that are large and, in particular, have high out degrees. We conjecture that for larger datasets, the running time spent on computing the 99% fastest alignments will be significantly smaller than the running time spent on the 1% slowest alignments. Here, even faster methods for computing fragmentation tree alignments are sought. We will evaluate whether our ILP is capable of solving these ‘hard’ instances faster than a DP-based approach, as its running time is not directly dependent on the out degree of the trees.

We have put particular focus on fragmentation trees that are hard to align, namely large trees with high out degrees. Small trees with low out degree seem to be less interesting since they often belong to small compounds (<300 Da). Often, these compounds are ‘knowns’ (that is, reference measurements of the compound can be found in a spectral library) and can be identified by spectral comparison. Also, small fragmentation trees contain less information for, say, classifying an unknown compound. Nevertheless, we believe that we can also speed up alignments when one of the fragmentation trees is relatively small: this may be achieved using some preprocessing for small trees with, say, less than four losses.

We conjecture that running time of the DP (Theorems 1 and 2) can be improved to  $O(2^{d_1+d_2} \cdot \text{poly}(d_1, d_2) n_1 n_2)$  using the Möbius transform (Björklund *et al.*, 2007), but this appears to be of theoretical interest only.

In our evaluations, we have used a scoring function similar to the one by Rasche *et al.* (2012). Both scorings lack any statistical explanation and should be refined in the future using, say, log odds scores. Also, the effect of merging two or possibly even more nodes has to be investigated. Both questions were beyond the scope of this work. Another interesting question is whether polynomial-time methods for tree alignment of unordered trees, such as the constrained tree edit distance (Zhang, 1996), can be used for aligning fragmentation trees: whereas the restrictions imposed by Zhang (1996) have no sensible interpretation in the context of fragmentation trees, quality of results may still be sufficient for certain applications.

Aligning fragmentation trees allows for an automated classification of unknown compounds into compound classes. Thus, large-scale compound screens can easily be searched for compounds of interest. This may be useful in the search for signaling molecules, biomarkers, or novel drugs and the identification of illegal drugs or toxins. In conjunction with other methods from



systems biology, the concept can help to identify new metabolic pathways based on tandem MS experiments.

## ACKNOWLEDGMENTS

We thank Aleš Svatoš (MPI for Chemical Ecology, Jena, Germany), Masanori Arita (University of Tokyo, Japan) and David Grant and Dennis Hill (University of Connecticut, Storrs, USA) for providing the MS datasets.

*Funding:* International Max Planck Research School Jena [stipend to F.H.]; and the Carl-Zeiss-Foundation [to M.C.].

*Conflict of Interests:* Value of two patents may be affected by publication (S.B., F.H. and F.R.).

## REFERENCES

- Arora,S. *et al.* (1998). Proof verification and the hardness of approximation problems. *J. ACM*, **45**, 501–555.
- Backofen,R. *et al.* (2011). Sparse RNA folding: time and space efficient algorithms. *J. Discrete Algorithms*, **9**, 12–31.
- Björklund,A. *et al.* (2007). Fourier meets Möbius: fast subset convolution. In *Proceedings of ACM Symposium on Theory of Computing (STOC 2007)*, ACM Press, New York, pp. 67–74.
- Böcker,S. and Rasche,F. (2008). Towards de novo identification of metabolites by analyzing tandem mass spectra. *Bioinformatics*, **24**, 149–155. [*Proceedings of European Conference on Computational Biology (ECCB 2008)*].
- Canzar,S. *et al.* (2011). On tree-constrained matchings and generalizations. In *Proceedings of International Conference on Automata, Languages and Programming (ICALP 2011)*, Springer, Berlin, Vol. 6755, pp. 98–109.
- Cui,Q. *et al.* (2008). Metabolite identification via the Madison Metabolomics Consortium Database. *Nat. Biotechnol.*, **26**, 162–164.
- Fernie,A.R.*et al.* (2004). Metabolite profiling: from diagnostics to systems biology. *Nat. Rev. Mol. Cell Biol.*, **5**, 763–769.
- Fiehn,O. (2008). Extending the breadth of metabolite profiling by gas chromatography coupled to mass spectrometry. *Trends Analyt. Chem.*, **27**, 261–269.
- Halket,J.M. *et al.* (2005). Chemical derivatization and mass spectral libraries in metabolic profiling by GC/MS and LC/MS/MS. *J. Exp. Bot.*, **56**, 219–243.
- Herlihy,M. *et al.* (2008). Hopscotch hashing. In *Proceedings of Symposium on Distributed Computing (DISC 2008)*, Springer, Berlin, Vol. 5218, pp. 350–364.
- Hill,D.W. *et al.* (2008). Mass spectral metabonomics beyond elemental formula: chemical database querying by matching experimental with computational fragmentation spectra. *Anal. Chem.*, **80**, 5574–5582.
- Horai,H. *et al.* (2010). MassBank: a public repository for sharing mass spectral data for life sciences. *J. Mass Spectrom.*, **45**, 703–714.
- Jiang,T. *et al.* (1995). Alignment of trees: an alternative to tree edit. *Theor. Comput. Sci.*, **143**, 137–148.
- Last,R.L. *et al.* (2007). Towards the plant metabolome and beyond. *Nat. Rev. Mol. Cell Biol.*, **8**, 167–174.
- Lederberg,J. (1965). Topological mapping of organic molecules. *Proc. Natl. Acad. Sci. USA*, **53**, 134–139.
- Le,S.Y. *et al.* (1989). Tree graphs of RNA secondary structures and their comparisons. *Comput. Biomed. Res.*, **22**, 461–473.
- Li,J.W.-H. and Vederas, J.C. (2009). Drug discovery and natural products: end of an era or an endless frontier? *Science*, **325**, 161–165.
- Ljubić,I. *et al.* (2005). Solving the prize-collecting steiner tree problem to optimality. In *Proceedings of Algorithm Engineering and Experiments (ALENEX 2005)*, SIAM, pp. 68–76.
- Neumann, S. and Böcker, S. (2010). Computational mass spectrometry for metabolomics – a review. *Anal. Bioanal. Chem.*, **398**, 2779–2788.
- Oberacher,H. *et al.* (2009). On the inter-instrument and inter-laboratory transferability of a tandem mass spectral reference library: 1. results of an Austrian multicenter study. *J. Mass Spectrom.*, **44**, 485–493.
- Pagh,R. and Rodler, F.F. (2004). Cuckoo hashing. *J. Algorithms*, **51**, 122–144.
- Rasche,F. *et al.* (2011). Computing fragmentation trees from tandem mass spectrometry data. *Anal. Chem.*, **83**, 1243–1251.
- Rasche,F. *et al.* (2012). Identifying the unknowns by aligning fragmentation trees. *Anal. Chem.*, **84**, 3417–3426.
- Rauf,I. *et al.* (2012). Finding maximum colorful subtrees in practice. In *Proceedings of Research in Computational Molecular Biology (RECOMB 2012)*. Springer, Berlin, Vol. 7262, pp. 213–223.
- Scheubert,K. *et al.* (2011). Computing fragmentation trees from metabolite multiple mass spectrometry data. In *Proceedings of Research in Computational Molecular Biology (RECOMB 2011)*, Vol. 6577, pp. 377–391.
- Schmidt,B.M. *et al.* (2007). Revisiting the ancient concept of botanical therapeutics. *Nat. Chem. Biol.*, **3**, 360–366.
- Sniedovich,M. (2006). Dijkstra’s algorithm revisited: the dynamic programming connexion. *Control Cybern.*, **35**, 599–620.
- Werner,E. *et al.* (2008). Mass spectrometry for the identification of the discriminating signals from metabolomics: current status and future trends. *J. Chromatogr. B*, **871**, 143–163.
- Zhang,K. and Jiang,T. (1994). Some MAX SNP-hard results concerning unordered labeled trees. *Inf. Process. Lett.*, **49**, 249–254.
- Zhang,K. (1996). A constrained edit distance between unordered labeled trees. *Algorithmica*, **15**, 205–222.