Speedy Colorful Subtrees

W. Timothy J. White [1], Stephan Beyer [2], Kai Dührkop [1], Markus Chimani [2] and Sebastian Böcker [1]

[1]Chair for Bioinformatics, Friedrich-Schiller-University, Jena, Germany,
`{tim.white,kai.duehrkop,sebastian.boecker}@uni-jena.de`
[2]Institute of Computer Science, University of Osnabrück, Germany,
`{stephan.beyer,markus.chimani}@uni-osnabrueck.de`

This is a preprint of the article: W. Timothy J. White, Stephan Beyer, Kai Dührkop, Markus Chimani and Sebastian Böcker. Speedy Colorful Subtrees. In Proc. of Computing and Combinatorics Conference (COCOON 2015), volume 9198 of Lect Notes Comput Sci, pages 310–322. Springer, Berlin, 2015.

Abstract. Fragmentation trees are a technique for identifying molecular formulas and deriving some chemical properties of metabolites—small organic molecules—solely from mass spectral data. Computing these trees involves finding exact solutions to the NP-hard Maximum Colorful Subtree problem. Existing solvers struggle to solve the large instances involved fast enough to keep up with instrument throughput, and their performance remains a hindrance to adoption in practice.
We attack this problem on two fronts: by combining fast and effective reduction algorithms with a strong integer linear program (ILP) formulation of the problem, we achieve overall speedups of 9.4 fold and 8.8 fold on two sets of real-world problems—without sacrificing optimality. Both approaches are, to our knowledge, the first of their kind for this problem. We also evaluate the strategy of solving *global* problem instances, instead of first subdividing them into many *candidate* instances as has been done in the past. Software (C++ source for our reduction program and our CPLEX/Gurobi driver program) available under LGPL at `https://github.com/wtwhite/speedy_colorful_subtrees/`.

1 Introduction

Metabolites—small molecules involved in cellular reactions—provide a direct functional signature of cellular state. Untargeted metabolomics aims to identify all such compounds present in a biological or environmental sample, and the predominant technology in use is mass spectrometry (MS). This remains a challenging problem, in particular for the many compounds that cannot be found in any spectral library [17, 18]. Here we consider tandem mass spectra (MS^2), which measure the masses and abundances of fragments of an isolated compound.

A first step toward full structural elucidation of a compound is the identification of its molecular formula. While it is possible to derive the molecular formula for a given exact mass, measurement inaccuracies have to be considered. Even for high-accuracy instruments, when using an appropriate error range for the mass measurement there may be thousands of possible molecular formulas for a given mass [7]. Approaches for identifying the correct formula include isotope pattern analysis [3], fragmentation pattern analysis [2], or a combination of both [8, 9, 11, 12, 15].

Computation of fragmentation trees [12] is a highly powerful method for fragmentation pattern analysis: In the 2013 CASMI (Critical Assessment of Small Molecule Identification) Challenge for identifying molecular formulas, a combination of fragmentation tree and isotope pattern analysis was selected "best automated tool" [6, 10]. In addition, fragmentation tree structure can help to derive information about an unknown compound's structure [13, 16]. Peaks in the spectrum are annotated with molecular formulas by looking for *consistent explanations*, using knowledge of possible fragmentation events and their probabilities. This translates into finding exact solutions to the NP-hard Maximum Colorful Subtree (MCS) problem, described later. Unfortunately the problem instances generated can contain over 100,000 edges, and the performance of existing approaches cannot keep up with the throughput of the MS instruments, sometimes limiting the method's appeal in practice. Heuristics often fail to find the optimal solution, and a simple integer linear program (ILP) has been identified as the fastest exact method [14].

We attack this problem on two fronts: by combining fast and effective reduction algorithms with facet-defining inequalities for the ILP formulation of the problem, we achieve overall speedups of 9.4 fold and 8.8 fold on two sets of real-world problems—without sacrificing optimality. Both approaches are, to our knowledge, the first of their kind for this problem. We also evaluate the strategy of solving *global* problem instances, instead of first subdividing them into many *candidate* instances as has been done in the past. Here, we will not evaluate the quality of solutions, as these are identical for any exact method; also, we will assume the edge weights of the MCS problem to be given [5].

1.1 Fragmentation Trees Are Maximum Colorful Subtrees

Consider an MS^2 spectrum containing k peaks $p_1, \ldots, p_k$, having mass-to-charge (m/z) ratios m_i and peak intensities q_i for $1 \leq i \leq k$, listed in decreasing m/z order. Following Böcker and Rasche [2], we use the Round-Robin algorithm [1] to find all possible *explanations* of the parent peak—that is, all candidate molecular formulas having m/z approximately equal to m_1. Each such formula becomes the 1-colored root vertex in a separate MCS instance graph. Within each MCS instance, i-colored vertices are added for each possible explanation of peak p_i, for all $2 \leq i \leq k$. Whenever the molecular formula of v is a subformula of the formula of u, indicating that v could possibly be generated by fragmenting u, we add a directed edge (u, v) and assign an edge weight (which may be positive, negative or zero) according to a probabilistic model of fragmentation. Intuitively,

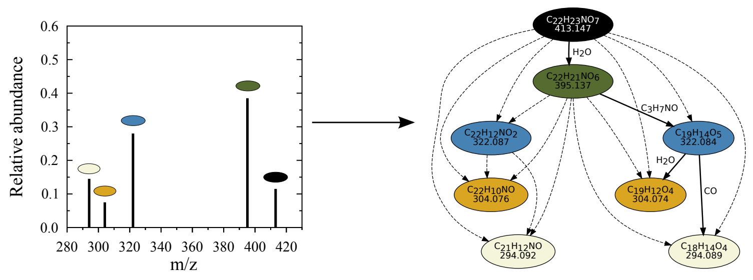

Relative abundance
0.0
0.1
0.2
0.3
0.4
0.5
0.6
280 300 320 340 360 380 400 420
m/z
$C_{22}H_{23}NO_7$ 413.147
H_2O
$C_{22}H_{21}NO_6$ 395.137
C_3H_7NO
$C_{22}H_{12}NO_2$ 322.087
$C_{19}H_{14}O_5$ 322.084
H_2O
CO
$C_{22}H_{10}NO$ 304.076
$C_{19}H_{12}O_4$ 304.074
$C_{21}H_{12}NO$ 294.092
$C_{18}H_{14}O_4$ 294.089

Fig. 1. Example MS^2 spectrum and resulting MCS instance. Nodes (peak explanations) show their molecular formulas and weights. Edges (fragmentation events) in the optimal subtree are solid and labeled with their neutral losses. Edge weights not shown.

a rooted colorful subtree T in one of these graphs maps each peak to at most one molecular formula in such a way that all formulas in T are consistent with fragmentation of the candidate formula at the root, with the tree of highest total weight corresponding to the best such explanation. By calculating the weights of these optimal trees for all MCS instances and ranking them, the best candidate formula for the spectrum can be determined. Fig. 1 shows an example.

The full version of this paper discusses a technique for solving a single *global* MCS instance representing the entire problem, instead of multiple *candidate* instances.

Formally, an instance of the MCS problem is given by (V, E, C, w, c, r) where V is the set of vertices, $E \subsetneq V^2$ is the set of directed edges, C is the set of colors, $w\colon E \to \mathbb{R}$ is the weight function on edges, $c\colon V \to C$ is the function defining colors for each vertex, and $r \in V$ is a distinguished vertex called the *root*. The graph (V, E) is acyclic, and there is a path from r to every $v \in V$.

A subgraph $G' \subseteq G$ is *colorful* iff all vertices in G' have different colors. The weight of an edge $e = (u, v) \in E$ is given by $w(u, v)$, and we define $w(u, v) = -\infty$ when $(u, v) \notin E$. We further extend this function to operate on any subgraph G' in the usual way, by summing over all edges in G'. A subgraph $X \subseteq G$ *dominates* a subgraph $Y \subseteq G$ iff $w(X) \geq w(Y)$.

We would like to assign weights to both edges and vertices: the former to reflect the likelihood of the specific neutral loss in question; the latter to capture peak-specific or explanation-specific information such as peak intensity, mass deviation between measurement and prediction, and estimates of formula plausibility. In order to represent a weight function $w'\colon V \cup E \to \mathbb{R}$ on both vertices and edges using a weight function $w\colon E \to \mathbb{R}$ on edges only, we can simply set $w(u, v) := w'(u, v) + w'(v)$ for each $(u, v) \in E$, since every valid subtree containing v must contain exactly one incoming edge (u, v).

Our goal is to find a maximum colorful r-rooted subtree T of G: that is, among all subtrees rooted at r and in which at most one vertex of any given color appears, a subtree having maximum total weight. This problem is NP-hard. It remains NP-hard even if G is a tree with unit edge weights [2], or if color constraints are dropped [14].[1]

We say that a subgraph G' is *below* a vertex u iff there is a path of zero or more edges from u to every vertex in G'. We denote by G_u the unique maximal subgraph of G below u. A color i is below u iff there exists a path from u to a vertex of color i. Furthermore a color i is below a color i' iff there exists an i'-colored vertex u such that i is below u. A subtree T of a graph G is *full in* G iff it is rooted at some vertex u of G, and every edge in G below u is in T. We use the term "cost" to describe a (typically negative) quantity that is to be *added* to a weight to produce another weight. We also declare a vertex to be an ancestor of itself, and use the adjective *strict* to denote non-self ancestors.

Let $n := |V|$, $m := |E|$, and $k := |C|$. Let $\delta^-(U) := \{(v,u) \mid u \in U, v \notin U\}$, $\delta^+(U) := \{(u,v) \mid u \in U, v \notin U\}$, and $\delta(U) := \delta^-(U) \cup \delta^+(U)$. When $U = \{u\}$ we dispense with the braces. We also define $V_i := \{v \in V \mid c(v) = i\}$.

2 Data Reduction

Our data reduction rules seek to shrink an MCS instance X to a smaller instance X' by deleting edges that are provably unnecessary—that is, edges that are simultaneously absent from some optimal solution to X. Here we outline our rules and their computationally efficient implementations.

Vertex upper bounds. The following sections describe upper bounds $U(\cdot)$ on the maximum-weight subtree rooted at some vertex u. Particular upper bounds are named by subscripting U; when just U with no subscript appears, it means that any arbitrary upper bound can be substituted. Trivially we have that $U(u) \geq 0$ for all u, since the 0-weight tree containing just u and no edges is a subtree rooted at u. A computationally useful property of all vertex upper bounds is that they remain valid in the face of edge deletions, enabling reductions to safely delete multiple edges in between bound updates.

Child upper bound. A simple upper bound $U_\chi(\cdot)$ for a given vertex u can be obtained by considering the upper bounds of u's children and the edges leading from u to them. Specifically we may choose, among all u's outgoing edges to i-colored children $(u,v) \in \delta^+(u) \cap \delta^-(V_i)$, either the edge (u,v) that maximises $w(u,v) + U(v)$ or no edge if this expression is negative. Summing across colors i yields equation (1). This bound tends to become very loose for vertices near the top of the graph, since high-weight edges near the bottom of the graph will usually be visited by large numbers of paths. Nevertheless it is capable of eliminating many edges near the bottom of the graph when applied to the vertex upper bound reduction rule. It can be considerably strengthened

[1] When edge weights are constrained to be nonnegative and color constraints are dropped, all leaves will appear in some optimal solution and the problem reduces to the polynomial-time-solvable maximum spanning tree problem.

by incorporating other vertex upper bounds, such as the Colorful Forest upper bound.

$$U_\chi(u) = \sum_{i \in \{c(v) | (u,v) \in E\}} \max\Big\{0, \max_{\substack{(u,v)\in E,\\ c(v)=i}} \big(w(u,v) + U(v)\big)\Big\} \tag{1}$$

We calculate this bound in $O(m \log k)$ time and $O(n+k)$ space using dynamic programming.

Colorful Forest upper bound. We next describe an upper bound $U_\lambda(\cdot)$ obtained by relaxing the subtree constraint. Consider a vertex u, and the subgraph G_u below u. Suppose that for each color i below u we choose either no edge, or some edge $(v, v') \in E(G_u)$ such that $c(v') = i$. All colorful forests in G_u may be generated by choosing incoming edges in this way, and this set of subgraphs contains the set of all colorful subtrees rooted at u, so the problem of finding a maximum-weight colorful forest in G_u is a relaxation of the u-rooted MCS problem. The optimal solution to the relaxed problem is easily found by choosing, for each color i, the maximum-weight incoming edge when this is positive and no edge otherwise, yielding an upper bound on the weight of a colorful subtree rooted at u. This is given in equation (2).

$$U_\lambda(u) = \sum_{i \in C} \max\Big\{0, \max_{\substack{(v,v')\in E(G_u),\\ c(v')=i}} w(v, v')\Big\} \tag{2}$$

Dynamic programming permits calculation in $O(km)$ time and $O(kn)$ space.

Strengthening the Colorful Forest bound. The Colorful Forest bound can be strengthened by noticing that whenever the forest that it constructs fails to be a tree, we can determine an upper bound on the cost that must be incurred to transform it into one. This upper bound can be added to the weight of the forest to produce a new, stronger vertex upper bound $U_\Lambda(\cdot)$. Here we merely mention that careful implementation allows this stronger bound to be computed in the same time complexity as the original; for a full description, see the full version of this paper.

Anchor lower bound. Given that a vertex u is in the solution T, what is a lower bound $L_a(u, v)$ on the cost of forcing in a given vertex v? Here we assume that T does not already contain a $c(v)$-colored vertex, and only consider attaching v to a vertex in T by a single edge.

If v is a child of u, then clearly $w(u, v)$ is a possibility. Regardless, it may still be possible to attach v to a strict ancestor of u. Specifically, since the "anchor" vertex u is in T by assumption, either $u = r$ or one of the parents of u is also in T. Therefore to form a lower bound, we have the option of attaching v to u if this is possible, or to the worst of u's parents, recursively:

$$L_a(u, v) = \begin{cases} \max\left\{w(u, v), \min_{(p,u)\in E} L_a(p, v)\right\}, & u \neq r \\ w(r, v), & u = r \end{cases}$$

recalling that we define $w(u,v) = -\infty$ whenever $(u,v) \notin E$. ($L_a(u,v)$ will produce $-\infty$ iff there is some path from r to u that contains no vertex with an edge to v.)

$L_a(\cdot,\cdot)$ can be computed via dynamic programming in $O(n^2)$ time and space.

It is also helpful to define $L_{a'}(u,v) := \min_{(p,u)\in E} L_a(p,v)$. This variant of $L_a(\cdot,\cdot)$ excludes any direct edge from u to v from consideration.

Slide lower bound. Suppose we have a solution T which contains a vertex u. We want to calculate a lower bound $L_s(u,v)$ on the cost of changing T into a new solution T' by replacing u with another given vertex v of the same color as u. We call this the *Slide lower bound* because in the usual representation of fragmentation graphs, vertices of the same color occupy the same row, so forcing v into and u out of T is akin to horizontally sliding the endpoint of an edge in T from u to v. Such a modification may in general completely change the vertices and edges in the tree below u, subject to the important restriction that it respects color usage: that is, it only ever transforms a subtree T_u into a subtree T'_v such that $c(T'_v) \subseteq c(T_u)$. This reflects the fact that we cannot safely insert vertices of new colors, because these colors may already be in use by other parts of the solution. The full version of this paper describes how to compute $L_s(u,v)$ by dynamic programming in $O(mn_k)$ time and space, where n_k is the maximum number of vertices of any color.

Vertex Upper Bound rule. If for some edge (u,v) we have that $w(u,v) + U(v) \leq 0$, then clearly any solution containing (u,v) is dominated by a solution in which (u,v) and any subtree below it have been deleted, implying that (u,v) can be safely deleted. Applying this rule before other rules removes certain uninteresting special cases from consideration.

Slide rule. Whenever two edges exist from a vertex u to distinct vertices v and v' of the same color, there is an opportunity to apply the Slide reduction rule. If

$$w(u,v') - w(u,v) + L_s(v,v') > 0 \tag{3}$$

holds then any solution T containing (u,v) can be improved by sliding (u,v) to (u,v'). This rule can be strengthened by replacing the first term with $L_a(u,v')$, which affords us the chance to connect v' to an ancestor of u. We may then usefully allow $v' = v$ to eliminate edges (u,v) that can always be replaced with a better edge (a,v), where a is a strict ancestor of u.[2]

Dominating Path rule. The idea behind the Slide rule can be taken further: instead of trying to replace an edge (u,v) with another single edge from an ancestor of u to a vertex of the same color as v, we can replace it with a chain of

[2] The full version of this paper discusses a subtlety regarding floating-point arithmetic and comparisons for equality.

d edges connecting vertices $v_1, \ldots, v_{d+1}$, with the starting point v_1 an ancestor of u and the endpoint v_{d+1} obeying $c(v_{d+1}) = c(v)$ as before. However we must now pay a price for forcing in each internal vertex v_j for all $2 \leq j \leq d$ in this chain, because the solution may already contain some different vertex of color $c(v_j)$ that needs to be dealt with. This can be done for each such internal vertex by using the Slide lower bound. Suppose the path we wish to force in contains some i-colored vertex x, but the solution already contains a conflicting vertex—an i-colored vertex $y \neq x$. The solution can be patched up by deleting the incoming edge to y and sliding any subtree below y so that it appears below x for a total cost of $L_s(y, x) - w(p, y)$, where p is y's parent in the solution. Since we do not know, for any color i, which i-colored vertex (if any) is already in the solution, we must take the worst case over all i-colored vertices and all their possible incoming edges:

$$L_{\text{force}}(x) = \min_{y \in V_{c(x)}} \Big(L_s(y, x) - \max_{(p,y) \in E} w(p, y) \Big) \tag{4}$$

It is now possible to state a recursion to calculate an upper bound on the cost to force in a given vertex x, assuming that a vertex u is already in the solution:

$$f(u, x) = \min \Big\{ 0, \alpha, \alpha + L_{\text{force}}(x) \Big\} \tag{5}$$

$$\alpha = \max \Big\{ L_a(u, x), \max_{p,\, (p,x) \in E} \big(f(u, p) + w(p, x) \big) \Big\} \tag{6}$$

We now have that an edge (u, v) can be deleted if there exists an edge (x, z) such that $c(z) = c(v)$ and $w(x, z) + f(u, x) + L_s(v, z) > w(u, v)$. $f(u, x)$ can be calculated in $\mathrm{O}(n)$ space because its first argument never varies during recursion.

Two further reduction rules, the Implied Edge rule and the Color Combining rule, are described in the full version of this paper.

3 Integer Linear Programming

Rauf *et al.* [14] surveyed different methods to obtain optimal solutions of the MCS problem, including an integer linear program (ILP). We extend this to obtain a strictly stronger LP relaxation, and solve the resulting ILP using the cutting plane method.

The ILP of Rauf *et al.* [14] is equivalent to

$$\max \quad \sum_{(u,v) \in E} w(u, v)\, x_e \tag{7a}$$

$$\text{s.t.} \quad \sum_{e \in \delta^-(V_i)} x_e \leq 1 \qquad \forall i \in C, \tag{7b}$$

$$\sum_{e \in \delta^-(v)} x_e \geq x_{(v,u)} \qquad \forall (v, u) \in E, v \neq r \tag{7c}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \tag{7d}$$

where x_e is assigned 1 iff the directed edge e is included in the solution. For each $v \in V \setminus \{r\}$ their formulation also includes a constraint $\sum_{e\in\delta^-(v)} x_e \leq 1$, but these constraints are redundant due to the *colorful forest constraints* (7b), which ensure that every color is contained in the solution at most once and that there is at most one incoming directed edge for each vertex. The *connectivity constraints* (7c) say that for each non-root vertex of V, there may only be outgoing directed edges if there is an incoming directed edge. Note that the ILP has a linear number of constraints and variables, so its linear relaxation can be solved as-is without separation.

In this paper, we add the constraints

$$\sum_{e\in\delta^-(V_i)} x_e \leq \sum_{f\in\delta^-(S)} x_f \quad \forall i \in C \quad \forall S \subseteq V, V_i \subseteq S \tag{8}$$

that prohibit splits and joins of fractional values. The constraints are valid for the ILP since they only forbid the case where the left-hand side is 1 and the right-hand side is 0, which could only happen if the result is not connected. However, the constraints make the LP relaxation strictly stronger, as can be seen in Fig. 2: in Fig. 2(b) the incoming value of v_1 is 0.5 and the incoming value of v_3 is 1 which is forbidden by (8) for $S = \{v_1, v_2, v_3\}$ and $i = c(v_3)$.

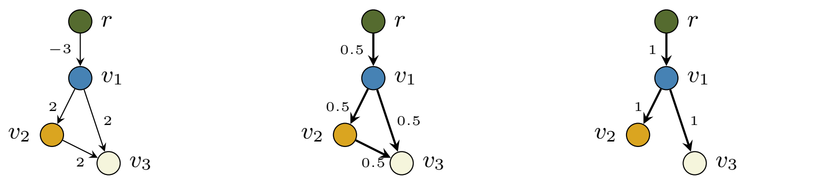
We first solve the LP for a subset of the constraints. Then, we solve the *separation problem*: we search (8) for one or more violated constraints, add them to the LP, and re-solve, iterating the process until there are no further violated constraints. Here, the separation problem can be answered by finding, for each $i \in C$, a minimum r-V_i-cut in the solution network (V, E, x) and testing if it is less than $\sum_{e\in\delta^-(V_i)} x_e$. Although (8) contains an exponential number of constraints, the separation problem can be solved in polynomial time using a Maximum Flow algorithm, and only a small number of iterations are typically needed to find a feasible LP solution.

Theorem 1. (7b) *and* (8) *provide facet-defining inequalities of the problem polytope and are the only necessary ones.*

The proof is given in the full version of this paper. Although just these inequalities suffice for correctness, we also keep (7c) for evaluation in practice because they do not need to be separated.

4 Results and Discussion

We tested the performance of our reductions and ILP improvements on a spectral dataset containing 1232 compounds that appear in the KEGG `http://www.kegg.jp` metabolite database. From this we selected hard instances where the "classic" ILP from Rauf *et al.* [14]—previously being the fastest exact method for the MCS problem—showed poor running times. We computed fragmentation graphs for each compound and built two datasets for evaluation:



(a) the input graph, each vertex has a different color (b) optimal solution of LP relaxation of (7) with objective value 1.5 (c) optimal solution of LP relaxation of (7)$\wedge$(8) with objective value 1

Fig. 2. An example showing that the LP relaxation of (7) including (8) is strictly stronger than without (8).

– graphs100: A set containing the 10 hardest candidate instances as well as a random sample of a further 90 hard candidate instances. We use this dataset to measure the performance of our reductions and ILP improvements.
– fmm1: A set of 20 hard global instances, comprising 86358 candidate instances in total. We use this dataset to compare the heretofore typical strategy of solving all candidate MCS instances separately, to solving a single global instance. Results for this dataset are given in the full version of this paper.

Rauf *et al.* [14] found that 95 % of MCS instances could be solved by ILP in under 5 seconds, while some took up to 5.6 minutes. To this end, it is sufficient to consider the hard instances in our comparison. The full version of this paper describes both the datasets and our results in more detail.

We implemented our reductions in `ft_reduce`, a C++ program that understands a simple language for describing the sequence of reductions to perform, affording flexibility in testing different orders and combinations of reductions. We selected three representative reduction scripts to analyse:

– R1 computes vertex upper bounds using both the Child bound and the Colorful Forest bound, and then applies the Vertex Upper Bound rule.
– R2 does the same, but uses the strengthened Colorful Forest bound.
– R3 applies R2 and then all remaining reduction rules.

Each script iterates until no more edges can be removed. The full version of this paper gives the complete scripts.

We implemented our new ILP formulation using a C++ driver program linked with CPLEX 12.6.0 (`http://www.ibm.com/software/integration/optimization/cplex-optimization-studio/`). Our new facet-defining cuts can be turned on or off using a command-line argument. In the remainder, we call the solver with these cuts turned on "CPLEX+Cuts", and the solver with them turned off "CPLEX" or "stock CPLEX". For the separation of the split-and-join constraints, we use the Maximum Flow code by Cherkassky and Goldberg [4]. We also performed tests using Gurobi 5.5.0 (`http://www.gurobi.com/`), although we were not able to implement the cuts efficiently using its callback framework.

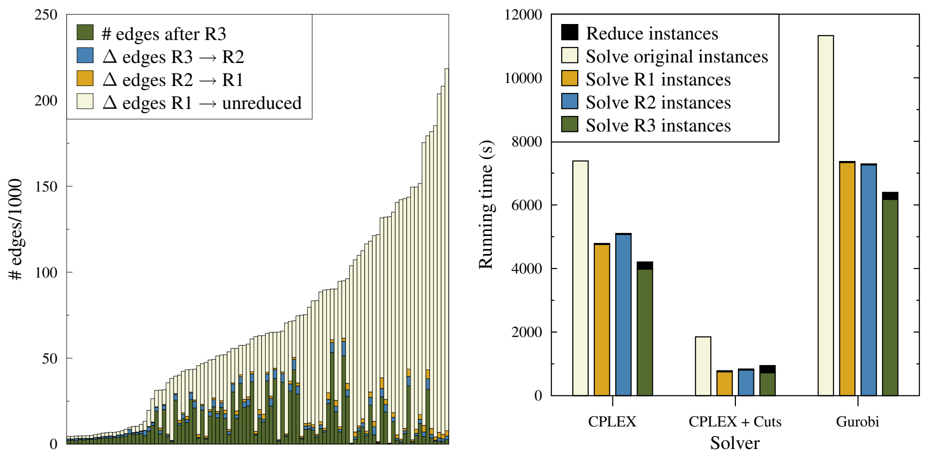


Fig. 3. Left: Comparing unreduced and reduced instance sizes for each graphs100 instance. The bottom bar in each stack gives the number of edges after R3 reduction; higher bars correspond to weaker reductions, with the entire stack indicating the unreduced instance size. Right: Running time evaluation for all graphs100 instances. Each column shows the total elapsed time needed to solve all 100 instances, with reduction time broken out as a black bar at the top.

All computational experiments were performed on a cluster of four 12-CPU 2.4GHz E5645 Linux machines with 48 GB RAM each. All reductions and all ILP solver runs for the graphs100 dataset ran to completion with a RAM limit of 4 GB and a time limit of 2 hours in place. For the fmm1 dataset, the memory limit was increased from 4 GB to 12 GB, but some instances failed to run to completion in the 2 hour limit. Our reduction program is single-threaded, and ILP solvers were operated in single-threaded mode. All time measurements are in elapsed (wallclock) seconds, and exclude time spent on I/O.

4.1 Results for graphs100 Dataset

Fig. 3 (left) shows the effectiveness of our reductions in shrinking the graphs100 problem instances. Every R1 or R2 reduction removed at least 11.6 % of the edges, and every R3 reduction removed at least 35.4 %, with the average reductions being 62.4 %, 64.3 % and 70.4 % for R1, R2 and R3, respectively. Many instances produced much larger reductions, and it is clear from Fig. 3 (left) that reduced instance size is only very weakly correlated with original instance size.

Fig. 3 (right) compares the performance of various combinations of reduction scripts and ILP solvers. Two effects are immediately apparent: using the strengthened ILP formulation improves average solution times for CPLEX by at least a factor of 4; and applying either the R1 or R2 reduction script produces anywhere from a 30.9 % decrease (from unreduced to R2 on stock CPLEX) to a 57.6 % decrease (from unreduced to R1 on CPLEX+Cuts). We note with particular interest that applying both techniques is substantially *more* effective than would be expected by performing each separately: assuming their effects on running time to be independent, we would expect that both performing an R1 reduction and changing from stock CPLEX to CPLEX+Cuts would result in instances taking on average $0.69109 * 0.25065 = 0.173$ times as long to solve, but in fact we find that they take only 0.106 times as long—a relative improvement of 38.7 %, representing a 9.42-fold overall reduction in execution time.

In the other direction, we observe that both stock CPLEX and CPLEX+Cuts take slightly longer to solve the R2 instances than the R1 instances, despite the fact that every R2 instance's edge set is a strict subset of the corresponding R1 instance's edge set, having on average 5.1 % fewer edges. We can only surmise that the additional edges removed by using the strengthened Colorful Forest bound destroyed some structure sought by CPLEX's various heuristics.

The more expensive R3 reductions are a net improvement for the stock CPLEX and Gurobi solvers, but result in an overall slowdown for CPLEX+Cuts.

The highest memory usage on any unreduced instance was 1166MB, 1199MB and 956MB for stock CPLEX, CPLEX+Cuts and Gurobi, respectively. On reduced instances these figures dropped to 853MB, 498MB and 640MB. The highest memory usage by our `ft_reduce` program was 15MB, 15MB and 31MB for R1, R2 and R3 reductions, respectively.

5 Conclusion

We have presented two highly effective techniques for accelerating the optimal solution of MCS instances, thereby bringing practical *de novo* identification of metabolite molecular formulas a step closer to reality. The two methods complement each other admirably: applying both yields a *larger* speedup than the product of the speedups obtained by applying each separately. Based on our experiments with two real-world datasets, we find that it is essentially always advantageous to use our strengthened ILP formulation and to apply our simple reductions, and frequently advantageous to apply our more complex ones.

The lion's share of the improvement in running times comes from our new, facet-defining cutting planes for ILP solvers. ILP solvers have demonstrated effectiveness across a wide range of hard optimization problems, and we anticipate that they will remain the dominant approach to solving MCS problems. At the same time, the problem reductions we present offer immediately-available speedups (and, often, memory usage reductions) not only for ILP formulations but for any exact or heuristic solution method, such as the "brute force" algorithm of Böcker and Rasche [2] or the Tree Completion heuristic of Rauf *et al.* [14].

We noted above that the use of fragmentation trees goes beyond the determination of molecular formulas [13]: see for instance Shen *et al.* [16] where fragmentation trees are used in conjunction with machine learning to search a molecular structure database using fragmentation spectra. In this analysis pipeline, computing fragmentation trees accounts for more than 90 % of the total running time. To this end, faster methods for this task are highly sought.

Bibliography

[1] Böcker, S. and Lipták, Zs. (2007). A fast and simple algorithm for the Money Changing Problem. *Algorithmica*, **48**(4), 413–432.

[2] Böcker, S. and Rasche, F. (2008). Towards de novo identification of metabolites by analyzing tandem mass spectra. *Bioinformatics*, **24**, I49–I55. Proc. of *European Conference on Computational Biology* (ECCB 2008).

[3] Böcker, S., Letzel, M., Lipták, Zs., and Pervukhin, A. (2009). SIRIUS: Decomposing isotope patterns for metabolite identification. *Bioinformatics*, **25**(2), 218–224.

[4] Cherkassky, B. and Goldberg, A. (1997). On implementing push-relabel method for the maximum flow problem. *Algorithmica*, **19**, 390–410.

[5] Dührkop, K. and Böcker, S. (2014). Fragmentation trees reloaded. In *Proc. of Research in Computational Molecular Biology (RECOMB 2015)*. Accepted for publication.

[6] Dührkop, K., Hufsky, F., and Böcker, S. (2014). Molecular formula identification using isotope pattern analysis and calculation of fragmentation trees. *Mass Spectrom*, **3**(special issue 2), S0037.

[7] Kind, T. and Fiehn, O. (2006). Metabolomic database annotations via query of elemental compositions: Mass accuracy is insufficient even at less than 1 ppm. *BMC Bioinformatics*, **7**(1), 234.

[8] Menikarachchi, L. C., Cawley, S., Hill, D. W., Hall, L. M., Hall, L., Lai, S., Wilder, J., and Grant, D. F. (2012). MolFind: A software package enabling HPLC/MS-based identification of unknown chemical structures. *Anal Chem*, **84**(21), 9388–9394.

[9] Meringer, M., Reinker, S., Zhang, J., and Muller, A. (2011). MS/MS data improves automated determination of molecular formulas by mass spectrometry. *MATCH-Commun Math Co*, **65**, 259–290.

[10] Nishioka, T., Kasama, T., Kinumi, T., Makabe, H., Matsuda, F., Miura, D., Miyashita, M., Nakamura, T., Tanaka, K., and Yamamoto, A. (2014). Winners of CASMI2013: Automated tools and challenge data. *Mass Spectrom*, **3**(special issue 2), S0039.

[11] Pluskal, T., Uehara, T., and Yanagida, M. (2012). Highly accurate chemical formula prediction tool utilizing high-resolution mass spectra, MS/MS fragmentation, heuristic rules, and isotope pattern matching. *Anal Chem*, **84**(10), 4396–4403.

[12] Rasche, F., Svatoš, A., Maddula, R. K., Böttcher, C., and Böcker, S. (2011). Computing fragmentation trees from tandem mass spectrometry data. *Anal Chem*, **83**(4), 1243–1251.

[13] Rasche, F., Scheubert, K., Hufsky, F., Zichner, T., Kai, M., Svatoš, A., and Böcker, S. (2012). Identifying the unknowns by aligning fragmentation trees. *Anal Chem*, **84**(7), 3417–3426.

[14] Rauf, I., Rasche, F., Nicolas, F., and Böcker, S. (2013). Finding maximum colorful subtrees in practice. *J Comput Biol*, **20**(4), 1–11.

[15] Rojas-Chertó, M., Kasper, P. T., Willighagen, E. L., Vreeken, R. J., Hankemeier, T., and Reijmers, T. H. (2011). Elemental composition determination based on MS^n. *Bioinformatics*, **27**, 2376–2383.

[16] Shen, H., Dührkop, K., Böcker, S., and Rousu, J. (2014). Metabolite identification through multiple kernel learning on fragmentation trees. *Bioinformatics*, **30**(12), i157–i164. Proc. of *Intelligent Systems for Molecular Biology* (ISMB 2014).

[17] Tautenhahn, R., Cho, K., Uritboonthai, W., Zhu, Z., Patti, G. J., and Siuzdak, G. (2012). An accelerated workflow for untargeted metabolomics using the METLIN database. *Nat Biotechnol*, **30**(9), 826–828.

[18] Wishart, D. S., Knox, C., Guo, A. C., Eisner, R., Young, N., Gautam, B., Hau, D. D., Psychogios, N., Dong, E., Bouatra, S., Mandal, R., Sinelnikov, I., Xia, J., Jia, L., Cruz, J. A., Lim, E., Sobsey, C. A., Shrivastava, S., Huang, P., Liu, P., Fang, L., Peng, J., Fradette, R., Cheng, D., Tzur, D., Clements, M., Lewis, A., Souza, A. D., Zuniga, A., Dawe, M., Xiong, Y., Clive, D., Greiner, R., Nazyrova, A., Shaykhutdinov, R., Li, L., Vogel, H. J., and Forsythe, I. (2009). HMDB: A knowledgebase for the human metabolome. *Nucleic Acids Res*, **37**, D603–D610.