Complexity of Splits Reconstruction for Low-Degree Trees*

Serge Gaspers[†] Mathieu Liedloff[‡] Maya Stein[§] Karol Suchan[¶‖]

Abstract

Given a vertex-weighted tree T, the split of an edge xy in T is $\min\{s_x(xy), s_y(xy)\}$ where $s_u(uv)$ is the sum of all weights of vertices that are closer to u than to v in T. Given a set of weighted vertices V and a multiset of splits $\mathcal{S}$, we consider the problem of constructing a tree on V whose splits correspond to $\mathcal{S}$. The problem is known to be NP-complete, even when all vertices have unit weight and the maximum vertex degree of T is required to be no more than 4. We show that

- the problem is strongly NP-complete when T is required to be a path,
- the problem is NP-complete when all vertices have unit weight and the maximum degree of T is required to be no more than 3, and
- it remains NP-complete when all vertices have unit weight and T is required to be a caterpillar with unbounded hair length and maximum degree at most 3.

We also design polynomial time algorithms for

- the variant where T is required to be a path and the number of distinct vertex weights is constant, and
- the variant where all vertices have unit weight and T has a constant number of leaves.

The latter algorithm is not only polynomial when the number of leaves, k, is a constant, but also fixed-parameter tractable when parameterized by k.

Finally, we shortly discuss the problem when the vertex weights are not given but can be freely chosen by an algorithm.

The considered problem is related to building libraries of chemical compounds used for drug design and discovery. In these inverse problems, the goal is to generate chemical compounds having desired structural properties, as there is a strong correlation between structural properties, such as the Wiener index, which is closely connected to the considered problem, and biological activity.

1 Introduction

In this paper, we consider trees $T = (V, E)$ where integer weights are associated to vertices by a function $\omega : V \to \mathbb{N}$, where $\mathbb{N}$ denotes the set of natural numbers excluding 0.

Definition 1. *Let T be a tree and $\omega : V \to \mathbb{N}$ be a function. The* split *of an edge e in T is the minimum of $\Omega(T_1)$ and $\Omega(T_2)$, where T_1 and T_2 are the two trees obtained by deleting e from T, and $\Omega(T_i) = \sum_{v \in T_i} \omega(v)$. We use $\mathcal{S}(T)$ to denote the multiset of splits of T.*

We consider the problem of reconstructing a tree with a given multiset of splits and a given set of weighted vertices.

*A preliminary version of this article appeared in the proceedings of WG 2011 [10]. The authors acknowledge the support of Conicyt Chile via projects Fondecyt 11090390 (M.L., K.S.), Fondecyt 11090141 (M.S.), Anillo ACT88 (K.S.), and Basal-CMM (S.G., M.S., K.S.). The first author acknowledges partial support from the European Research Council (COMPLEX REASON, 239962). The second and fourth authors acknowledge the support of the French Agence Nationale de la Recherche (ANR AGAPE ANR-09-BLAN-0159-03)

[†]Institute of Information Systems, Vienna University of Technology, Vienna, Austria. E-mail: gaspers@kr.tuwien.ac.at

[‡]LIFO, Université d'Orléans, Orléans, France. E-mail: mathieu.liedloff@univ-orleans.fr

[§]CMM, Universidad de Chile, Santiago, Chile. E-mail: mstein@dim.uchile.cl

[¶]FIC, Universidad Adolfo Ibáñez, Santiago, Chile. E-mail: karol.suchan@uai.cl

[‖]WMS, AGH - University of Science and Technology, Krakow, Poland.

WEIGHTED SPLITS RECONSTRUCTION (WSR): Given a set V of n vertices, a weight function $\omega : V \to \mathbb{N}$, and a multiset $\mathcal{S}$ of integers, is there a tree T whose multiset of splits is $\mathcal{S}$ (i.e. $\mathcal{S}(T) = \mathcal{S}$)?

The WEIGHTED SPLITS RECONSTRUCTION FOR TREES OF MAXIMUM DEGREE k problem (WSR_k) is defined in the same way, except that we restrict the tree T to have maximum degree at most k. When we require T to belong to a class of trees $\mathcal{T}$, the problem is called WEIGHTED SPLITS RECONSTRUCTION FOR $\mathcal{T}$.

When ω assigns unit weights to the vertices, the problem is simply called SPLITS RECONSTRUCTION (SR). The SPLITS RECONSTRUCTION FOR TREES OF MAXIMUM DEGREE k problem (SR_k) and the SPLITS RECONSTRUCTION FOR $\mathcal{T}$ are the obvious unweighted counterparts of the weighted variants defined above.

Related Work. In the field of Chemical Graph Theory [2, 3, 19], molecules are modeled by graphs in order to study the physical properties of chemical compounds. A chemical graph is a graph, where vertices represent atoms of a chemical compound and edges the chemical bonds between them. Within the area of quantitative structure-activity relationship (QSAR), several structural measures of chemical graphs were identified that quantitatively correlate with a well defined process, such as biological activity or chemical reactivity. Probably the most widely known example is the *Wiener index* (see [13]): the sum of the distances in a graph between each pair of vertices, where the distance between two vertices is the length (the number of edges) of a shortest path from one to the other. Wiener [20] found a strong correlation between the boiling points of paraffins and the Wiener index. From then on, many other topological (using the information of the chemical graph) and topographical (using the information of the chemical graph and the location of its vertices in space) indices were introduced and their correlation with various other biological activities was investigated.

In Combinatorial Chemistry, drug design is facilitated by building libraries of molecules that are structurally related (via the Wiener index or any of the other numerous indices). We face inverse problems where the goal is to design new compounds that have a prescribed structural information (see also [6]).

Goldman et al. [12] study problems related to the design of combinatorial libraries for drug design from an algorithmic and complexity-theoretic point of view, following the heuristic approaches of [18] and [11]. Goldman et al. show that for every positive integer W, except 2 and 5, there exists a graph with Wiener index W. They also show that every integer, except a finite set, is the Wiener index of some tree. For constructing a tree (of unbounded or bounded maximum degree) with a given Wiener index, they devise pseudo-polynomial dynamic programming algorithms. Goldman et al. also introduce the SPLITS RECONSTRUCTION problem and recall a result due to Wiener [20]: the Wiener index of a tree T on n vertices with unit weights is $\sum_{s \in \mathcal{S}(T)} s \cdot (n - s)$. They show that SR is NP-complete and give an exponential-time algorithm without running time analysis.

As it is not reasonable to construct chemical trees with arbitrarily high vertex degrees, Li and Zhang [16] studied SR_4 and showed that it is also NP-complete. Their algorithm to construct a tree with maximum degree at most 4 to solve SR_4 runs in exponential time (no running time analysis is provided) and creates weighted vertices in intermediate steps.

In order to reconstruct glycans or carbohydrate sugar chains, Aoki-Kinoshita et al. [1] study the reconstruction of a node-labeled supertree from a set of node-labeled subtrees. They give a 6-approximation algorithm for this problem, which generalizes the smallest superstring problem.

We refer to [4] surveying results on the Wiener index for trees.

Our Results. By the result of Li and Zhang [16], SR_4 is NP-complete, while SR_2 is trivially in P. We close this gap by showing that SR_3 is NP-complete by a reduction from NUMERICAL MATCHING WITH TARGET SUMS (defined below). It is even NP-complete for caterpillars with unbounded hair length. Identifying small classes of trees for which the problem is NP-complete may be important for future investigations in the spirit of the deconstruction of hardness proofs [15] which aim at identifying parameters for which the problem becomes tractable when these parameters are small.

Our main result proves that WSR_2 is strongly NP-complete by a reduction from a variant of NUMERICAL MATCHING WITH TARGET SUMS in which all integers of the input are distinct. For the case where the weights of the vertices are chosen from a small set of values, our dynamic-programming algorithm solves WSR_2 in time $O(n^{k+3} \cdot k)$, where k is the number of distinct vertex weights. Although this running time is polynomial for every constant k, the degree of the polynomial depends on k. Thus, the

running time becomes impractical, even for small values of k. Parameterized complexity [5, 8, 17] is a theoretical framework that allows to distinguish between running times of the form $f(k)n^{g(k)}$ where the degree of the polynomial depends on the parameter k and running times of the form $f(k)n^{O(1)}$ where the exponential explosion of the running time is restricted to the parameter only. The fundamental hierarchy of parameterized complexity classes is

$$\text{FPT} \subseteq \text{W[1]} \subseteq \text{W[2]} \cdots \subseteq \text{XP},$$

where a parameterized problem is in FPT (fixed-parameter tractable) if there is a function f such that the problem can be solved in time $f(k)n^{O(1)}$, a problem is in XP if there are functions f, g such that the problem can be solved in time $f(k)n^{g(k)}$, and W[t], $t \geq 1$, are parameterized intractibility classes giving strong evidence that a parameterized problem that is hard for any of these classes is not in FPT. Our algorithm for WSR_2 parameterized by the number of distinct vertex weights places this problem in XP. A generalization of this problem is W[1]-hard [7], but it remains open whether this problem is fixed parameter tractable. As a relevant parameter for SR we identified k, the number of leaves in the reconstructed tree. This parameterization of SR can be solved in time $O(8^{k \log k} \cdot n)$, and is thus fixed-parameter tractable.

Definitions. A *caterpillar* is a tree consisting of a path, called its *backbone*, and paths attached with one end to the backbone. Its *hair length* is the maximum distance (in terms of the number of edges) from a leaf to the closest vertex of the backbone. A *star* $K_{1,k}$ is a tree with k leaves and one internal vertex, called the *center*. In our hardness proofs, we reduce from the following problem (problem [SP17] in [9]).

> Numerical Matching with Target Sums (NMTS): Given three disjoint multisets A,B, and $S = \{s_1, \ldots, s_m\}$, each containing m elements from $\mathbb{N}$, can $A \cup B$ be partitioned into m disjoint sets $C_1, C_2, \ldots, C_m$, each containing exactly one element from each of A and B, such that, for $1 \leq i \leq m$, $\sum_{c \in C_i} c = s_i$?

Organization. The remainder of this paper is organized as follows. Section 2 shows that WSR_2 is NP-complete. On the positive side, we show in Section 3 that WSR_2 can be solved in polynomial time when the number of distinct vertex weights is bounded by a constant. That this result cannot be extended to WSR_3 is shown in Section 4: SR_3 remains NP-complete. Section 5 gives an FPT-algorithm for SR parameterized by the number of leaves of the reconstructed tree. The variant where the vertex weights are freely choosable is discussed in Section 6 and we conclude with some directions for future research in Section 7.

2 WSR_2 is strongly NP-complete

In this section, we show that WSR_2 is strongly NP-complete. First we introduce a new problem that is polynomial-time-reducible to WSR_2, and then show that this new problem is strongly NP-hard.

> Scheduling with Common Deadlines (SCD): Given n jobs with positive integer lengths $j_1, \ldots, j_n$ and n deadlines $d_1 \leq \ldots \leq d_n$, can the jobs be scheduled on two processors P_1 and P_2 such that at each deadline there is a processor that finishes a job exactly at this time, and processors are never idle between the execution of two jobs?

To reinforce the intuition on this problem one may imagine that we want to satisfy delivery deadlines and avoid using any warehouse space to store a product between its fabrication and the delivery date. There is no restriction as to which product should be delivered at a given time. (Another possibility is imagining computer scientists scheduling paper production to fit conference deadlines.)

Given an instance $(j_1, \ldots, j_n, d_1, \ldots, d_n)$ for SCD, we construct an instance for WSR_2 as follows. For each job j_i, $1 \leq i \leq n$, create a vertex v_i with weight $\omega(v_i) = j_i$. For each deadline d_i, $1 \leq i \leq n-1$, create a split d_i. We may assume that $\sum_{i=1}^{n} j_i = d_{n-1} + d_n$, otherwise we trivially face a No-instance.

Suppose the path $P = (v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)})$ is a solution to WSR_2. Say $\{v_{\pi(\ell)}, v_{\pi(\ell+1)}\}$ is the edge associated to the split d_{n-1}. We construct a solution for SCD by assigning the jobs $j_{\pi(1)}, j_{\pi(2)}, \ldots, j_{\pi(\ell)}$ to processor P_1, and the jobs $j_{\pi(n)}, j_{\pi(n-1)}, \ldots, j_{\pi(\ell+2)}, j_{\pi(\ell+1)}$ to processor P_2, in this order. Note that

then, one of the jobs $j_{\pi(\ell)}$, $j_{\pi(\ell+1)}$ ends at d_{n-1}, and the other at $-d_{n-1} + \sum_{i=1}^{n} j_i = d_n$, which is as desired.

On the other hand, if SCD has a solution, then WSR_2 has a solution as well, because the previous construction is easily inverted. Visually, the list of jobs of P_2 is reversed and appended to the list of jobs of P_1. Job lengths correspond to vertex weights and deadlines correspond to splits (the last deadline where a job from P_1 finishes is merged with the last deadline where a job from P_2 finishes). Thus, SCD is polynomial-time-reducible to WSR_2.

Lemma 2. $\mathrm{SCD} \leq_p \mathrm{WSR}_2$.

In the remainder of this section, we show that dNMTS is polynomial-time-reducible to SCD. The dNMTS problem is equal to the NMTS problem, except that all integers in $A \cup B \cup S$ are pairwise distinct. This variant has been shown to be strongly NP-hard by Hulett et al. [14]. As the proof becomes somewhat simpler, we use dNMTS instead of NMTS for our reduction.

Let us first give a high level description of the main ideas of the reduction. For a dNMTS instance (A, B, S), the elements of $A \cup B$ will be encoded as jobs, and the elements of S will be encoded as deadlines. A convenient way to represent an element $s \in S$ is by introducing segments which are delimited to the left and the right by double deadlines, and whose distance is equivalent to s. The elements of $A \cup B \cup S$ are blown up by well-chosen additive factors that preserve solutions and make sure that the length of each segment can only be met by the sum of exactly two job-lengths, one corresponding to an element of A and the other to an element of B.

Our reduction will create an instance whose solution assigns, in each segment, one x-job (a job corresponding to an A-element) and one y-job (a job corresponding to a B-element) to the same processor, such that these two jobs are the only jobs executed on this processor in this segment, thus providing a solution to dNMTS. Without loss of generality, the x-job is scheduled first. As we must not introduce any restriction which x-jobs can be assigned to which segments, we introduce a deadline for each length of an x-job; these are the real deadlines. We refer to the x- and y-jobs as green jobs. The job lengths were blown up such that in each segment, exactly one processor starts with a green x-job, and in each segment, exactly one processor ends by executing a green y-job. In each segment, the green jobs must not overlap; this is achieved by multiplying all deadlines created so far and the corresponding job lengths by a factor 2, and introducing fake deadlines at odd positions one unit before the real deadlines. If an x-job and a y-job overlapped, there would be no job ending at the fake deadline preceding the real deadline at which the x-job ends, as all green jobs have even length and all real deadlines and double deadlines are even. Blue, red, and black jobs are created to meet all deadlines on the processor that is not currently executing green jobs. The blow-up of the elements of $A \cup B \cup S$ ensures that these jobs cannot equate the green jobs (except the black jobs whose lengths might equal the lengths of green y-jobs, but, without loss of generality, one can assign them to the last part of each segment of the processor not executing a green job). That none of these jobs is executed between two green jobs within a segment is ensured as the sum of all green job lengths equals the sum of the lengths of the segments. This summarizes the reduction and gives the reasons for the different elements of the construction. Let us now turn to the formal reduction.

Let (A, B, S) be an instance for dNMTS. We suppose, without loss of generality, that $\sum_{i=1}^{m} s_i = \sum_{x \in A \cup B} x$, otherwise (A, B, S) is trivially a No-instance for dNMTS. Let $A = \{a_1, \ldots, a_m\}$ and $B = \{b_1, \ldots, b_m\}$. We also assume, without loss of generality, that $a_i < a_{i+1}$, $b_i < b_{i+1}$, $s_i < s_{i+1}$, for all $i \in \{1, \ldots, m-1\}$, that $a_m < b_m$, and that $s_m \leq a_m + b_m$.

First, we construct an equivalent instance (X, Y, Z) for dNMTS. Each of $X := \{x_1, \ldots, x_n\}$, $Y := \{y_1, \ldots, y_n\}$, and $Z := \{z_1, \ldots, z_n\}$ has $n := m + 1$ elements:

$$\begin{aligned}
&\text{for } i \in \{1, \ldots, n-1\}, \\
&x_i := 2 \cdot (a_i + (b_m + 2)), && x_n := 2 \cdot (a_m + 1 + (b_m + 2)), \\
&y_i := 2 \cdot (b_i + 3 \cdot (b_m + 2)), && y_n := 2 \cdot (b_m + 1 + 3 \cdot (b_m + 2)), \\
&z_i := 2 \cdot (s_i + 4 \cdot (b_m + 2)), \text{ and} && z_n := 2 \cdot (a_m + b_m + 2 + 4 \cdot (b_m + 2)).
\end{aligned}$$

The elements of X, Y, and Z have the following properties.

Property 1. *Each element of $X \cup Y \cup Z$ is an even positive integer.*

Property 2. *For every $i \in \{1, \ldots, n-1\}$, we have that $x_i < x_{i+1}$, that $y_i < y_{i+1}$, and that $z_i < z_{i+1}$.*

Property 3. *For every $i \in \{1, \ldots, n\}$, we have*

$$\begin{aligned} 2 \cdot b_m + 4 \leq x_i \leq 4 \cdot b_m + 4, \\ 6 \cdot b_m + 12 \leq y_i \leq 8 \cdot b_m + 14, \text{ and} \\ 8 \cdot b_m + 16 \leq z_i \leq 12 \cdot b_m + 18. \end{aligned}$$

In particular, Property 3 implies that $y_1 > x_n$, that $z_1 > y_n$, and that $2 \cdot y_1 > z_n$. Properties 1–3 easily follow by construction of X, Y, and Z.

Property 4. *If k and ℓ are integers such that $x_k + y_\ell = z_n$, then $k = \ell = n$.*

Property 4 holds because x_n and y_n are the only elements of X and Y, respectively, that are large enough to sum to z_n.

Property 5. *Let $p, q \in X \cup Y$, $p \leq q$, and $z \in Z$. If $p + q = z$, then $p \in X$ and $q \in Y$.*

By Property 3, the sum of any two X-elements is smaller and the sum of any two Y-elements is larger than any element of Z.

For our SCD instance, we create the following deadlines:

- *real* deadlines: $r_{i,j} := x_i + \sum_{k=1}^{j} z_k$, for each $j \in \{0, \ldots, n-1\}$ and each $i \in \{1, \ldots, n\}$,
- *fake* deadlines: $f_{i,j} := r_{i,j} - 1$, for each $j \in \{0, \ldots, n-1\}$ and each $i \in \{1, \ldots, n\}$, and
- *sum* deadlines: two deadlines $ds_{1,j} := ds_{2,j} := \sum_{k=1}^{j} z_k$, for each $j \in \{1, \ldots, n\}$.

The sum deadlines we just defined partition the interval $[0, ds_{1,n}]$ into n segments $I_j := [ds_{1,j-1}, ds_{1,j}]$, $j = 1, \ldots n$, where for convenience, we let $ds_{1,0} = 0$. We create jobs with the following lengths, where $x_0 = 0$:

- green x-jobs: x_i, for each $i \in \{1, \ldots, n\}$,
- green y-jobs: y_i, for each $i \in \{1, \ldots, n\}$,
- blue jobs: $n \cdot (n-1)$ times a job of length 1,
- red fill jobs: $n-1$ times a job of length $x_i - 1 - x_{i-1}$, for each $i \in \{1, \ldots, n\}$,
- red overlap jobs: $x_i - x_{i-1}$, for each $i \in \{1, \ldots, n\}$,
- black fill jobs: $z_i - x_n$ for $i \in \{1, \ldots, n-1\}$, and
- a black overlap job: $z_n - x_n + 1$.

To illustrate these definitions, we start by showing that if we have a YES-instance (X, Y, Z) for dN-MTS, then we have an SCD YES-instance as well. Let $C_1, C_2, \ldots, C_n$ be n couples such that $C_j = \{x_{\pi_1(j)}, y_{\pi_2(j)}\}$ and $x_{\pi_1(j)} + y_{\pi_2(j)} = z_j$, $j \in \{1, \ldots, n\}$, for two permutations π_1 and π_2 of the set $\{1, \ldots, n\}$. We construct a solution for SCD. Let us construct the schedules for P_1 and P_2. For each $j \in \{1, \ldots, n-1\}$,

- assign the green x-job $x_{\pi_1(j)}$ to the interval $[ds_{1,j-1}, r_{\pi_1(j),j-1}]$ of P_1,
- assign the green y-job $y_{\pi_2(j)}$ to the interval $[r_{\pi_1(j),j-1}, ds_{1,j}]$ of P_1,
- assign a red fill job of length $x_1 - 1$ to the interval $[ds_{1,j-1}, f_{1,j-1}]$ of P_2,
- for every $i \in \{1, \ldots, n-1\} \setminus \pi_1(j)$, assign a red fill job of length $x_{i+1} - 1 - x_i$ to the interval $[r_{i,j-1}, f_{i+1,j-1}]$ of P_2,
- for every $i \in \{1, \ldots, n\} \setminus \pi_1(j)$, assign a blue job to the interval $[f_{i,j-1}, r_{i,j-1}]$ of P_2,
- assign a red overlap job of length $x_{\pi_1(j)+1} - x_{\pi_1(j)}$ to the interval $[f_{\pi_1(j),j-1}, f_{\pi_1(j)+1,j-1}]$ of P_2, and

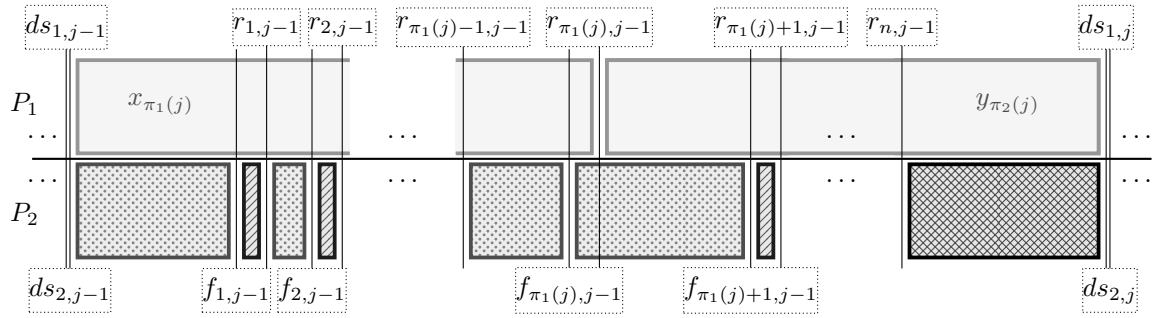

Figure 1: How jobs are assigned to processors in the SCD instance in segment $j < n$. The patterns of the jobs are as follows: Red jobs are dotted, blue jobs are hatched, black jobs are cross-hatched, and green jobs have no pattern.

- assign a black fill job of length $z_j - x_n$ to the interval $[r_{n,j-1}, ds_{1,j}]$ of P_2.

It only remains to assign jobs to the last segment. The last segment of P_1 contains the green x-job x_n and the green y-job y_n, in this order. The last segment of P_2 contains a red fill job of length $x_1 - 1$, a blue job, a red fill job of length $x_2 - 1 - x_1$, a blue job, ..., a red fill job of length $x_n - 1 - x_{n-1}$, and the black overlap job, in this order. See Fig. 1 for an illustration.

Now suppose the SCD instance is a YES-instance. We will show some structural properties of any valid assignment of jobs to the processors, which will help to extract a solution for our original dNMTS instance. We will show that in each segment I_j, any valid solution for the SCD instance has exactly one green x-job x_k and exactly one green y-job y_ℓ, and that x_k and y_ℓ sum to z_j.

Consider a valid assignment of the jobs to the processors P_1 and P_2. As two jobs with the same length are interchangeable, when we encounter a job whose length belongs to more than one category (for example "black fill" and "green y") we may choose in this case, without loss of generality, to which category the job belongs.

Claim 1. *A black fill job is assigned to each interval $[r_{n,j}, ds_{1,j+1}]$ with $j \in \{0, \dots, n-2\}$.*

Proof. Let $j \in \{0, \dots, n-2\}$. Two jobs must finish at the double deadline $ds_{1,j+1}, ds_{2,j+1}$. One of these must start at $r_{n,j}$ and thus has length $ds_{1,j+1} - r_{n,j} = \sum_{k=1}^{j+1} z_k - x_n - \sum_{k=1}^{j} z_k = z_{j+1} - x_n$. So this job is, without loss of generality, a black fill job. □

This uses up all black fill jobs.

Claim 2. *The green y-job y_n is assigned to the interval $[r_{n,n-1}, ds_{1,n}]$.*

Proof. As in the previous proof, one job must be assigned to this interval, whose length is $\sum_{k=1}^{n} z_k - x_n - \sum_{k=1}^{n-1} z_k = z_n - x_n$, which is y_n by Property 4. Thus, the green y-job y_n is assigned to the interval $[r_{n,n-1}, ds_{1,n}]$. □

Claim 3. *The black overlap job is assigned to the interval $[f_{n,n-1}, ds_{1,n}]$.*

Proof. As $r_{n,n-1}$ is the only deadline between $f_{n,n-1}$ and $ds_{1,n}$, the processor that does not use this deadline needs to process a job finishing at $ds_{1,n}$ and starting before $r_{n,n-1}$. This is the black overlap job, since no other job is long enough. It is assigned to the interval $[f_{n,n-1}, ds_{1,n}]$ of length $ds_{1,n} - f_{n,n-1} = z_n - x_n + 1$. □

This uses up all black jobs. Now, the only jobs left whose length is between $6b_m + 12$ and $8b_m + 14$ are the green y-jobs $y_1, \dots, y_{n-1}$.

Claim 4. *For each $\ell \in \{1, \dots, n-1\}$, the green y-job y_ℓ is assigned to an interval $[r_{i,j-1}, ds_{1,j}]$ for some $i \in \{1, \dots, n-1\}$ and $j \in \{1, \dots, n-1\}$.*

Proof. Each job is assigned to an interval inside some segment, as the double deadlines prevent jobs to span more than one segment. Suppose the green y-job y_ℓ is assigned to segment p. As $ds_{1,p} + y_\ell > ds_{1,p} + x_n$, by Properties 2 and 3, and the deadline following $r_{n,p} = ds_{1,p} + x_n$ is $ds_{1,p+1}$, it must be that the green y-job y_ℓ finishes at $ds_{1,p+1}$. Moreover, $ds_{1,p+1} - y_\ell$ is equal to a real deadline as $ds_{1,p+1} - y_\ell$ is even. □

Each of the $2n$ jobs that have been assigned so far finish at a double deadline $ds_{1,j}, ds_{2,j}$. Thus, no other jobs may end at a double deadline.

Claim 5. *A red fill job of length $x_1 - 1$ is assigned to each interval $[ds_{1,j}, f_{1,j}]$ with $0 \leq j \leq n-1$.*

Proof. Since both processors finish a job at deadline $ds_{1,j}$ (respectively, are initialized at time $ds_{1,0} = 0$) and one of them finishes a job at the following deadline, which is $f_{1,j}$, we need to assign a job of length $f_{1,j} - ds_{1,j} = x_1 - 1$ to the interval $[ds_{1,j}, f_{1,j}]$. Without loss of generality, this is one of the red fill jobs of length $x_1 - 1$. □

This uses up all red fill jobs of length $x_1 - 1$.

Claim 6. *For each $\ell \in \{1, \ldots, n\}$, the green x-job x_ℓ is assigned to an interval $[ds_{1,j}, r_{i,j}]$ for some $i \in \{1, \ldots, n\}$ and $j \in \{0, \ldots, n-1\}$.*

Proof. Suppose the green x-job x_ℓ is assigned to segment p. Notice that $x_\ell > r_{n,p} - f_{1,p}$. Indeed $r_{n,p} - f_{1,p} = x_n - x_1 + 1$ and, by construction, $x_n - x_1 + 1 \leq 2b_m$, whereas $x_\ell \geq 2b_m + 4$. Moreover, $r_{n,p}$ is the latest deadline in p. So the green x-job x_ℓ starts at $ds_{1,p}$. Notice that $ds_{1,p} + x_\ell < ds_{1,p+1}$ and that $ds_{1,p} + x_\ell$ corresponds to a real deadline as $ds_{1,p} + x_\ell$ is even, but all fake deadlines are odd. □

By Claims 2, 4, and 6, and since we have the same amount of segments as green x-jobs, respectively green y-jobs, we obtain that each segment I_j, $1 \leq j \leq n$,, contains exactly one green x-job and exactly one green y-job.

Claim 7. *For $j \in \{1, \ldots, n\}$, the green x-job and the green y-job in the segment I_j do not overlap.*

Proof. Suppose otherwise, that is, suppose there is a $j \in \{1, \ldots, n\}$ such that I_j contains a green x-job, say x_ℓ, and a green y-job, say y_k, that overlap (i.e. the intervals they are assigned to overlap). Since x_ℓ ends at a real deadline by Claim 6 and y_k starts at a real deadline by Claim 4, no job ends at the fake deadline situated at $ds_{1,j-1} + x_\ell - 1$, which contradicts the validity of the SCD solution. □

The last claim implies that in each segment I_j, $1 \leq j \leq n$, there is a green x-job x_{ℓ_j} and a green y-job y_{k_j} which together have the same size as the interval. Hence the couples $C_j = \{a_{\ell_j}, b_{k_j}\}, 1 \leq j \leq n$, form the desired solution of dNMTS. Thus, we have the following lemma.

Lemma 3. *dNMTS $\leq_p$* SCD.

We have assembled enough information to prove our main theorem.

Theorem 4. WSR_2 *is strongly* NP*-complete.*

Proof. The theorem follows from the strong NP-hardness of dNMTS, Lemmas 2 and 3, and the membership of WSR_2 in NP, which is easily verified as the certificate is a path and an assignment of the splits to its edges, all of which can be encoded in polynomial space. □

Corollary 5. Splits Reconstruction for Caterpillars of Unbounded Hair-Length and Maximum Degree 3 *is* NP*-complete.*

Proof. It is clear that this problem, abbreviated SRC, is in NP. To show that it is hard for NP, we reduce from WSR_2. Let $I'_P = (\omega'_1, \ldots, \omega'_{n-2}, s'_1, \ldots, s'_{n-3})$ be an instance of WSR_2, where ω'_i, $1 \leq i \leq n-2$, are the vertex weights and s'_j, $1 \leq j \leq n-3$, are the splits. We assume that all vertex weights and splits are upper bounded by a polynomial in n; as WSR_2 is strongly NP-hard, it is still NP-hard with this restriction. Define $\Omega := 1 + 2 \cdot \max\{\omega'_i : 1 \leq i \leq n-2\}$. To simplify the argument, consider an auxiliary instance $I_P = (\omega_1, \ldots, \omega_n, s_1, \ldots, s_{n-1})$ of WSR_2 obtained from I'_P by:

- augmenting the values of s'_j, $1 \leq j \leq n-3$, by Ω,

- adding $\omega_{n-1} = \omega_n = \Omega$ to the multiset of weights,
- adding $s_{n-2} = s_{n-1} = \Omega$ to the multiset of splits,
- and finally, multiplying each value in I_P by $n\Omega$ (so, for $1 \leq i \leq n-3$, $\omega_i = \omega'_i n\Omega$, and $s_i = (s'_i + \Omega)\Omega$).

It is not difficult to see that I_P and I'_P are equivalent.

Now let us create an instance I_C of SRC in the following way.

- replace each weight ω_i, $1 \leq i \leq n$, by ω_i copies of weight 1,
- for each ω_i, $1 \leq i \leq n$, add *auxiliary splits* $s_{f,i} = f$, $1 \leq f \leq \omega_i - 1$,
- keep the *original splits* $(s_1, \ldots, s_{n-1})$.

Notice that there are $\sum_{i=1}^{n} \omega_i$ vertices and $(\sum_{i=1}^{n} \omega_i) - 1$ splits (i.e. edges) in total.

As one easily checks, if I_P has a solution then I_C has a solution. Now suppose I_C has a solution C. Then, as C is an instance for SRC, it follows that C is a caterpillar of maximum degree 3 (with unbounded hair-length). Call B the backbone of C. Let $B' \subseteq B$ be maximal such that its endvertices have degree 3.

By construction $s_i > 1$, $1 \leq i \leq n-1$, and only the splits $s_{1,i}$, $1 \leq i \leq n$, have value 1. There are exactly n such splits, and so, C must have exactly n leaves.

Since there is no split of value $n\Omega^2 + 1$, each hair of C has length at most $n\Omega^2$. So, as $s_i > n\Omega^2$ for $i = 1, \ldots n-3$, we obtain that the splits $s_1, ..., s_{n-3}$ are assigned to edges $b_1, \ldots, b_{n-3}$ in $E(B')$. Observe that the edges $b_1, \ldots, b_{n-3}$ induce a connected graph (i.e. a path), as all other splits are smaller than the minimum of the s_i, $i = 1, \ldots n-3$.

Let P be the path formed by the edges $b_1, \ldots, b_{n-3}$ and let u and v be the two endpoints of P. There is at most one vertex y such that if we look at the values of the splits of the edges from u to y (resp. from v to y), then they are strictly increasing. In addition, if two edges of P share a vertex x, $x \neq y$, then there must be a hair attached to x, because the splits associated to these two edges differ by more than 1. Furthermore, there are hairs H_1 and H_2 of length $n\Omega^2$ attached to the first and to the last vertex on the backbone, as no two auxiliary splits are large enough to add up to one of the original splits s_i, $i = 1, \ldots n-3$. From the fact that C has exactly n leaves, it follows that the remaining hair has to be attached to y. As a consequence, $E(B') = \{b_1, \ldots, b_{n-3}\}$.

Let B'' be equal to B' augmented with the two edges to which the splits of value $n\Omega^2$ are assigned. All edges outside B'' (that is, edges from hairs) belong to auxiliary splits. This means that the edges adjacent to B'' correspond to auxiliary splits $s_{\omega_i - 1, i}$.

In order to find a solution for I_P, it thus suffices to take B'' and replace all hairs with the corresponding weight on their starting vertex on B''. □

3 Algorithm for WSR_2 with few distinct vertex weights

Let $k = |\{\omega(v) \ : \ v \in V\}|$ denote the number of distinct vertex weights in an instance $(V, \omega, \mathcal{S})$ for WSR_2. In this section, we exhibit a dynamic programming algorithm for WSR_2 that works in polynomial time when k is a constant. Moreover, standard backtracking can be used to actually construct a solution, if one exists.

Suppose $|V| = n$ and the multiset of splits, $\mathcal{S}$, contains the splits $s_1 \leq s_2 \ldots \leq s_{n-1}$. Let $w_1 < w_2 \ldots < w_k$ denote the distinct vertex weights and $m_1, m_2, \ldots, m_k$ denote their respective multiplicities, i.e. $m_i = |\{v \in V \ : \ \omega(v) = w_i\}|$ for all $i \in \{1, 2, \ldots, k\}$.

Our dynamic programming algorithm computes the entries of a boolean table A. The table A has an entry $A[p, W_L, W_R, v_1, v_2, \ldots, v_k]$ for each integer p with $1 \leq p \leq n-1$, each two integers $W_L, W_R \in \mathcal{S}$, and each $v_i \in \{0, 1, \ldots, m_i\}$, where $i \in \{1, 2, \ldots, k\}$. The entry $A[p, W_L, W_R, v_1, v_2, \ldots, v_k]$ is set to `true` iff there is an assignment of the splits $s_1, s_2, \ldots, s_p$ to the ℓ leftmost edges and the r rightmost edges of the path P_n on n vertices, such that

- $p = \ell + r$;

- v_1 weights w_1, v_2 weights w_2, ..., and v_k weights w_k are assigned to the ℓ leftmost and the r rightmost vertices of P_n such that each split assigned to the left (respectively to the right) part of the path corresponds to the sum of the vertex weights assigned to vertices to the left (respectively to the right) of this split; and
- W_L is equal to the value of the ℓ^{th} split from the left and W_R is equal to the r^{th} split from the right.

Intuitively, our algorithm assigns splits and weights by starting from both endpoints of the path and trying to join these two sub-solutions.

For the base case, set $A[0, W_L, W_R, v_1, v_2, \ldots, v_k]$ to `true` if $W_L = W_R = v_1 = v_2 = \ldots = v_k = 0$ and to `false` otherwise. We compute the remaining entries of A by increasing values of p using the following recurrence.

$$A[p, W_L, W_R, v_1, v_2, \ldots, v_k] = \bigvee_{i=1}^{k} \begin{cases} A[p-1, W_L - w_i, W_R, v_1, v_2, \ldots, v_{i-1}, \\ \quad v_i - 1, v_{i+1}, v_{i+2}, \ldots, v_k] \\ \vee A[p-1, W_L, W_R - w_i, v_1, v_2, \ldots, v_{i-1}, \\ \quad v_i - 1, v_{i+1}, v_{i+2}, \ldots, v_k] \end{cases}$$

In the previous recurrence, the formulas that refer to table entries that are undefined have the value `false`.

The final result of the algorithm is computed by evaluating the expression

$$\bigvee_{\substack{W_L, W_R \in \mathcal{S} \\ i \in \{1,2,\ldots,k\} \\ (W_L \leq w_i + W_R) \ \wedge \ (W_R \leq w_i + W_L)}} A[|\mathcal{S}|, W_L, W_R, m_1, m_2, \ldots, m_{i-1}, m_i - 1, m_{i+1}, m_{i+2}, \ldots, m_k].$$

Theorem 6. WSR_2 *can be solved in time* $O(n^{k+3} \cdot k)$*, where* k *is the number of distinct vertex weights of any input instance* $(V, \omega, \mathcal{S})$ *and* n *is the number of vertices.*

Proof. The correctness of the base case is clear. For the correctness of the recurrence, let v_{pivot} be the vertex on P_n where the two sub-solutions corresponding to the left and to the right part of P_n meet. First note that the values of the splits increase from left to right until we encounter vertex v_{pivot}, from which point they decrease. Filling up the path from both ends, this means that reading the splits from s_1 to s_{n-1}, we can assign them to the path, each time only deciding whether we assign it to the left part or to the right part of the path (in the SCD model, this would be equivalent to deciding whether to meet the next deadline on the processor P_1 or on the processor P_2). The first (respectively second) case of the recurrence corresponds to assigning the next split to the left (respectively right) part of the path by inserting a vertex of weight w_i, $i \in \{1, 2, \ldots, k\}$. The correctness of the final evaluation follows because it inserts the one missing vertex weight that has not been used between the left and the right part of the path.

The table has $|\mathcal{S}|^3 \cdot \Pi_{i=1}^{k}(m_i + 1) \leq n^{k+3}$ entries, each entry can be computed in time $O(k)$, and the final evaluation takes time $O(n \cdot k)$. □

4 SR_3 is NP-complete

In this section we show that Splits Reconstruction with unit weights is NP-complete for trees with maximum degree 3. Our polynomial-time reduction is done from the strongly NP-complete NMTS problem recalled in Section 1. This problem remains NP-complete even if each integer of the NMTS instance is at most $p(m)$, where p is a polynomial and m is the length of the description of the instance. Let us just mention that the next theorem does not immediately follow from Corollary 5.

Theorem 7. SR_3 *is* NP*-complete.*

Proof. Let $\tilde{A} = \{\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_m\}$, $\tilde{B} = \{\tilde{b}_1, \tilde{b}_2, \ldots, \tilde{b}_m\}$ and $\tilde{S} = \{\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_m\}$ be an instance of NMTS. Let $C = \max\{x \ : \ x \in \tilde{A} \cup \tilde{B}\}$. Without loss of generality, we construct the following equivalent NMTS

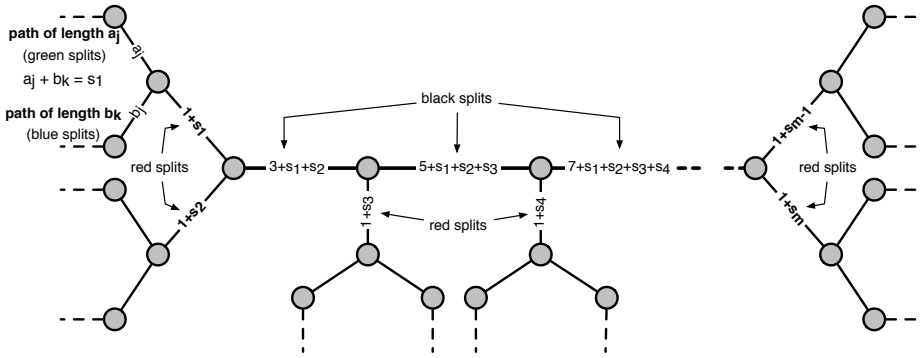

Figure 2: A tree with maximum degree 3 representing a solution to an SR_3 instance constructed as described in the proof of Theorem 7.

instance:

$$\begin{aligned} a_i &:= \tilde{a}_i + 2 + 3C, \quad 1 \le i \le m, \\ b_i &:= \tilde{b}_i + 3 + 5C, \quad 1 \le i \le m, \text{ and} \\ s_i &:= \tilde{s}_i + 5 + 8C, \quad 1 \le i \le m. \end{aligned}$$

Let $A = \bigcup_{1 \le i \le m} \{a_i\}$, $B = \bigcup_{1 \le i \le m} \{b_i\}$, and $S = \bigcup_{1 \le i \le m} \{s_i\}$. Clearly, the instance $(\tilde{A}, \tilde{B}, \tilde{S})$ has a solution if and only if the instance (A, B, S) has a solution.

Now we describe an instance $(V, \mathcal{S})$ of SR_3, which is a YES-instance if and only if the previous instance (A, B, S) of NMTS is a YES-instance (see also Figure 2).

Let $n = 2m - 2 + \sum_{i=1}^{m} a_i + \sum_{i=1}^{m} b_i$ be the number of vertices in the set V; we recall that these vertices have unit weight. The multiset $\mathcal{S}$ of splits is defined as follows.

- For each value s_i, $1 \le i \le m$, the value $1 + s_i$ is added to $\mathcal{S}$ and we refer to these splits as *red* splits.
- For each value s_i, $2 \le i \le m - 2$, the value $(i - 1) + \sum_{j=1}^{i} (1 + s_j)$ is added to $\mathcal{S}$ and we refer to these splits as *black* splits.
- For each value a_i, $1 \le i \le m$, the values $\{1, 2, \ldots, a_i\}$ are added to $\mathcal{S}$ and we refer to these splits as *green* splits.
- For each value b_i, $1 \le i \le m$, the values $\{1, 2, \ldots, b_i\}$ are added to $\mathcal{S}$ and we refer to these splits as *blue* splits.

Finally each value x of $\mathcal{S}$ is replaced by $\min(x, n - x)$. As required, $\mathcal{S}$ contains $n - 1$ splits.

Lemma 8. *(A, B, S) is a* YES*-instance for* NMTS *if and only if $(V, \omega : V \to \{1\}, \mathcal{S})$ is a* YES*-instance for* SR_3.

Proof. Throughout the proof, when we refer to a split of value x, we mean a split of value $\min(x, n - x)$.

"⇒" Assume that (A, B, S) is a YES-instance for NMTS. We will show that there is a solution to SR_3. A tree $T = (V, E)$ and a bijective function $b : E \to \mathcal{S}$ can be constructed as follows (see also Figure 2). Construct a path P with $m - 3$ edges with the black splits such that the $(i - 1)^{\text{th}}$ edge is associated to the black split $(i - 1) + \sum_{j=1}^{i} (1 + s_j), i \in \{2, 3, \ldots, m - 2\}$. Add two edges incident to the first vertex of P, that are associated to the red splits $1 + s_1$ and $1 + s_2$. Add two edges incident to the last vertex of P that are associated to the red splits $1 + s_{m-1}$ and $1 + s_m$. To the i^{th} vertex of P, $2 \le i \le m - 3$, add one incident edge associated to the red split $1 + s_{i+1}$. Finally, for each $a_i \in A$ and each $b_i \in B$, construct the paths with a_i and b_i vertices respectively. To each edge of these $2m$ paths, we can associate a green or a blue split. It remains to attach one green path and one blue path to each endpoint of an edge associated to a red split (one endpoint is already involved in the path P and of degree 3). The way to attach these path is given by the solution to the (A, B, S) instance.

"⇐" Assume that there exists a solution (T, b) to SR_3, where $T = (V, E)$ is a tree of maximum degree 3 and b is a bijection from E to $\mathcal{S}$. We show how a solution of the NMTS instance (A, B, S) can be derived from (T, b). Let us note that for any $i, j, k \in \{1, 2, \ldots, m\}$, we have that $a_i + s_j > s_k$, that $a_i + a_j < s_k$, that $b_i + b_j > s_k$, and that $a_i + a_j > b_k$.

Claim 8. *For every $i \in \{1, 2, \ldots, m\}$, there is a path on a_i edges, called the a_i-path, using the splits $1, 2, \ldots, a_i$ (without loss of generality, they are green) and there is a path on b_i edges, called the b_i-path, using the splits $1, 2, \ldots, b_i$ (without loss of generality, they are blue). All these a-paths and b-paths are edge-disjoint.*

Proof. As the instance has $2m$ splits of value 1, T has $2m$ leaves. Each of these leaves is incident to a green or blue split of value 1. As the instance also has $2m$ splits of each of the values $2, 3, \ldots, 2+3C$, the leaves of T are the starting points of $2m$ edge-disjoint paths $P_1, P_2, \ldots, P_{2m}$, each having $2 + 3C$ edges in T. Consider an $x \in A \cup B$ and the splits $2 + 3C + 1, 2 + 3C + 2, \ldots, x$. As $x < 4 + 6C$, and as there is no split smaller than $2 + 3C$ other than those we have already used to form the paths $P_i, 1 \leq i \leq 2m$, the splits $2 + 3C + 1, 2 + 3C + 2, \ldots, x$ are assigned to an extension of a path P_i, which, together with P_i, forms a path P'_i with x edges using the splits $1, 2, \ldots, x$. All these paths $P'_i, 1 \leq i \leq 2m$, are edge disjoint and without loss of generality, green splits are assigned to their edges if they have at most $2+4C$ edges and blue splits otherwise. □

Claim 9. *For every $i \in \{1, 2, \ldots, m\}$, the red split of value $1 + s_i$ is assigned to an edge e_i of T whose vertex u_i is the common extremity of an a-path and a b-path, where u_i is in the subtree of $T - e_i$ that has $s_i + 1$ vertices.*

Proof. As no split has value s_i, vertex u_i is incident to another two edges besides e_i. We note that all splits, besides those of the a- and b-paths, have value at least $6 + 8C$. One such split plus the smallest a_j, $1 \leq j \leq m$, would exceed s_i. So, u_i is the end point of two a/b-paths. These cannot be two a-paths as $a_j + a_k < s_i$, for any $j, k \in \{1, 2, \ldots, m\}$ and they cannot be two b-paths as $b_j + b_k > s_i$, for any $j, k \in \{1, 2, \ldots, m\}$. Thus, u_i is the common extremity of an a-path and a b-path. □

Finally, a solution to the instance (A, B, S) of NMTS is formed by the couples $C_1, C_2, \ldots, C_m$, where each C_i contains a_{i_a} and b_{i_b}, where i_a and i_b are such that the edge e_i of T, with $b(e_i) = 1 + s_i$, is incident to the a_{i_a}-path and the b_{i_b}-path. This proves the NP-hardness of SR_3. □

As the certificate is a tree on n vertices, the membership in NP is obvious and Theorem 7 is proved. □

5 Algorithm for SR with few leaves

In this section we design an algorithm for SR parameterized by the number k of splits that are equal to one, i.e. $k = |\{s = 1 : s \in \mathcal{S}\}|$. As each such split is incident to a leaf in a reconstructed tree, the algorithm reconstructs trees with k leaves.

The algorithm starts with a star T with center r and k leaves. The vertex r is also the root of T and r is the only vertex which is allowed to have non-unit weight during the execution of the algorithm. We start by setting $\omega(r) = n - k$. The splits that are equal to 1 are assigned to the edges of the star.

At any stage of the algorithm, T is a tree with splits from $\mathcal{S}$ assigned to its edges, and the goal is to replace the root r of T by a tree T_r with unit-weight vertices (except for the new root, that can have a non-unit weight), using splits from $\mathcal{S}$ that have not been assigned yet; the leaves of T_r are made adjacent to the former neighbors of r in T. If there exists such a replacement where the splits form a subset of $\mathcal{S}$, we say that T has a *valid extension*. Each tree T uniquely defines a partition (A, C, U) of the splits $\mathcal{S}$ such that

- A represents the multiset of *available* splits that have not yet been assigned to T,
- C represents the multiset of *current* splits assigned to edges incident to r, and
- U represents the multiset of *used* splits assigned to edges of T that are not incident to r.

Let b denote the value of the smallest split in C. Our tree T will grow out of r as follows.

- If $\omega(r) = 1$, then return TRUE. Indeed, T uses all splits from $\mathcal{S}$ as A is empty.

- If A contains a split whose value is at most b, then T has no valid extension and the algorithm backtracks. Indeed, on a path between two leaves in a valid tree, there is no split with value at most b between two splits with value at least b.
- If $|\{s \in A : s = b+1\}| > |\{s \in C : s = b\}|$, that is, A contains more splits with value $b+1$ than C contains splits with value b, then T has no valid extension and the algorithm backtracks. The correctness of this case holds by the pigeonhole principle and the argument used in the previous case.
- If $|\{s \in A : s = b+1\}| = |\{s \in C : s = b\}|$, then all valid extensions of T are also valid extensions of the tree obtained from T by subdividing each edge with split b that is incident to r. That is, for each edge rv with a split of value b, add a new vertex z_v, remove the edge rv, and add edges rz_v and $z_v v$. Decrement $\omega(r)$ by $|\{s \in C : s = b\}|$. The algorithm recursively solves the problem on this tree.
- Otherwise (if $|\{s \in A : s = b+1\}| < |\{s \in C : s = b\}|$), some split from C with value b receives a parent split with value more than $b+1$. Go over all choices for selecting a subset U of $N(r)$ of size at least 2 containing a vertex v such that rv is associated with a split with value b. If A contains no split that equals $1 + \sum_{u \in U} s(ru)$, where $s(e)$ denotes the split associated to the edge e of T, then discard this choice. Otherwise, create a new vertex z_U, remove the edges $\{ru : u \in U\}$ from T, add the edges $\{z_U u : u \in U \cup \{r\}\}$, and decrement $\omega(r)$ by 1. The algorithm resursively solves the resulting subproblems. If one such tree has a valid extension, T has a valid extension.

Theorem 9. SR *can be solved in time* $O(8^{k \log k} \cdot n)$*, where* $k = |\{s = 1 : s \in \mathcal{S}\}|$ *and* n *is the number of vertices.*

Proof. The arguments for correctness have been given in the description of the algorithm. For the running time analysis, we observe that $\omega(r)$ decreases in each recursive call, no recursive call increases $|C|$, and the time spent in each recursion step is linear. Let $T(c)$ denote an upper bound on the number of atomic instances solved for an instance with $|C| = c \leq k$, where an instance is atomic if the algorithm makes no recursive call for solving the instance. In the only case making more than one recursive call, we have

$$T(c) \leq \sum_{i=2}^{c} \binom{c}{i} T(c-i+1),$$

as the set U in the neighborhood of $N(r)$ is replaced by one vertex z_U. As $c \leq k$ and $\binom{c}{i} \leq k^i$, we have that

$$\begin{aligned} T(c) &\leq (c-1) \cdot \max_{i=2..c} \left\{ c^i \cdot T(c-(i-1)) \right\} \\ &\leq \max_{i=2..c} \left\{ k^{i+1} \cdot T(k-(i-1)) \right\} \\ &\leq \max_{i=2..c} \left\{ k^{(i+1)\frac{k}{i-1}} \right\} \ . \end{aligned}$$

This maximum is attained for $i = 2$, which proves the theorem. □

6 Freely choosable weights

We remark that the following modification of WSR makes any set of splits realizable in some tree. Suppose the weight function ω is not given, but freely choosable, that is, we ask whether, given a multiset $\mathcal{S}$ of integers, there exists a tree $T = (V, E)$ and a weight function $\omega : V \to \mathbb{N}$, such that $\mathcal{S}$ is the multiset of splits of T. We call this problem ChWSR.

Theorem 10. ChWSR *always admits a solution.*

Proof. We show that the answer to ChWSR is always yes: Decompose $\mathcal{S}$ into κ chains $s_1^j < s_2^j < \ldots s_{m(j)}^j$, $j = 1, \ldots, \kappa$, where κ is the maximal multiplicity in $\mathcal{S}$. Let T be obtained from the star $K_{1,\kappa}$ by subdividing e_j, the j^{th} edge of T, $m(j) - 1$ times (for $j = 1, \ldots, \kappa$), and root T at the center r of $K_{1,\kappa}$. Map s_i^j to the edges of the subdivided e_j, $1 \leq i \leq m(j)$, keeping their order, so that the edge

corresponding to s_1^j is incident to a leaf of T. Finally, choose the weight $\omega(r)$ for the root to be equal to the maximum value in $\mathcal{S}$. For each leaf v of T, set the weight $\omega(v)$ equal to the split assigned to the edge $\{v, u\}$, where u is the parent of v. Any other vertex v is given a weight equal to the difference of splits assigned to the edges incident to v. This choice of T and ω clearly satisfies the requirements. □

Remark. Due to the construction provided by the proof of Theorem 10, we note that we are not only always able to construct a tree T as required, but the structure of this tree is also rather simple. In particular, the realization of the split sequence is a path if each split in $\mathcal{S}$ repeats at most twice.

Observe that if we consider ChWSR with unit weights, we are back at the problem SR. It is not difficult to see that in SR, a given set of splits can be realized in the same way as explained in the proof of Theorem 10 for ChWSR, the only difference being that each time a non-unit weight w is assigned to some vertex v in ChWSR, in SR we have to add $w-1$ leaves of unit weight to v. Thus, if $\mathcal{S}$ contains a sufficient number of splits 1, then $\mathcal{S}$ can always be realized by a tree. More precisely, setting the boundary values $s_0^j := 0$ for all j, and letting κ denote the maximum multiplicity over all elements in $\mathcal{S}$ except 1, we have that if $\kappa \geq 2$ and $\mathcal{S}$ contains at least

$$\kappa + \sum_{j=1}^{\kappa} \sum_{i=1}^{m(j)} (s_i^j - s_{i-1}^j - 1) + 2 \cdot \max_{1 \leq i \leq \kappa} \{s_{m(i)}^i\} - 1 - \sum_{j=1}^{\kappa} s_{m(j)}^j$$

times the split 1, then $\mathcal{S}$ can be realized by a tree T: κ of them are needed to be assigned to edges incident to leaves of the star, $\sum_{i=1}^{m(j)}(s_i^j - s_{i-1}^j - 1)$ of them are added, with pending vertices, to vertices introduced by subdividing the edge e_j, and $2 \cdot \max_{i=1}^{\kappa}\{s_{m(i)}^i\} - 1 - \sum_{j=1}^{\kappa} s_{m(j)}^j$ of them are added, with pending vertices, to the root. (Note that it does not matter if there are more splits 1 than needed in our construction, since we may always add leaves to the center of T.) The previous condition is, of course, sufficient, but not necessary. Moreover, the tree T that realizes $\mathcal{S}$ is a subdivided star to which some leaves have been added. In particular, if each split in $\mathcal{S}$ repeats at most twice, then we can realize $\mathcal{S}$ in a caterpillar with hair-length one. We note that the conditions $\kappa = 2$ and the lower bound on the number of splits with value 1 are also necessary for caterpillars with hair length one.

7 Conclusion

In Section 3, we have shown that WSR_2 is in XP when parameterized by the number of distinct vertex weights. It remains open whether this problem is fixed parameter tractable (a generalization of the problem is W[1]-hard [7]). For practical purposes, it would further be important to identify other quantities that are small in practice (e.g. the diameter of the tree or topological indices), and investigate the multivariate complexity of the considered problems parameterized by combinations of these quantities.

There is a large contrast between the complexities of WSR, where we are given n vertex weights, and ChWSR, where we can freely choose the vertex weights, or, alternatively, we can choose the vertex weights from an infinite multiset containing n times each element of $\mathbb{N}$. It would be interesting to know some restrictions on the multiset of vertex weights such that the problem becomes polynomial time solvable, or fixed-parameter tractable with respect to interesting parameterizations, when we can chose the weights from this multiset. Ideally, these restrictions should be consistent with the applications in drug design and discovery.

Acknowledgment. We thank Ming-Yang Kao for communicating this problem.

References

[1] Kiyoko F. Aoki-Kinoshita, Minoru Kanehisa, Ming-Yang Kao, Xiang-Yang Li, and Weizhao Wang. A 6-approximation algorithm for computing smallest common aon-supertree with application to the reconstruction of glycan trees. In *Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC 2006)*, volume 4288 of *Lecture Notes in Computer Science*, pages 100–110. Springer, 2006.

[2] Alexandru T. Balaban. *Chemical Applications of Graph Theory.* Academic Press, Inc., 1976.

[3] Danail Bonchev and Dennis H. Rouvray. *Chemical Graph Theory: Introduction and Fundamentals.* Taylor & Francis, 1991.

[4] Andrey A. Dobrynin, Roger Entringer, and Ivan Gutman. Wiener index of trees: Theory and applications. *Acta Applicandae Mathematicae*, 66(3):211–249, 2001.

[5] Rodney G. Downey and Michael R. Fellows. *Parameterized complexity.* Springer-Verlag, 1999.

[6] Jean-Loup Faulon and Andreas Bender. *Handbook of Chemoinformatics Algorithms.* Chapman and Hall/CRC, 2010.

[7] Michael R. Fellows, Serge Gaspers, and Frances A. Rosamond. Parameterizing by the number of numbers. In *Proceedings of the 5th International Symposium on Parameterized and Exact Computation (IPEC 2010)*, volume 6478 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2010.

[8] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series.* Springer, Berlin, 2006.

[9] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[10] Serge Gaspers, Mathieu Liedloff, Maya J. Stein, and Karol Suchan. Complexity of splits reconstruction for low-degree trees. In *Proceedings of the 37th International Workshop on Graph-Theoretic Concepts in Computer Science WG 2011*, volume 6986 of *Lecture Notes in Computer Science.* Springer, 2011. To appear.

[11] Valerie J. Gillet, Peter Willett, John Bradshaw, and Darren V. S. Green. Selecting combinatorial libraries to optimize diversity and physical properties. *Journal of Chemical Information and Computer Sciences*, 39(1):169177, 1999.

[12] Deborah Goldman, Sorin Istrail, Giuseppe Lancia, Antonio Piccolboni, and Brian Walenz. Algorithmic strategies in combinatorial chemistry. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000)*, pages 275–284, 2000.

[13] Peter L. Hammer, editor. *Special issue on the 50th anniversary of the Wiener index, Discrete Applied Mathematics*, volume 80. Elsevier, 1997.

[14] Heather Hulett, Todd G. Will, and Gerhard J. Woeginger. Multigraph realizations of degree sequences: Maximization is easy, minimization is hard. *Operations Research Letters*, 36(5):594–596, 2008.

[15] Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. Deconstructing intractability - a multivariate complexity analysis of interval constrained coloring. *Journal of Discrete Algorithms*, 9(1):137–151, 2011.

[16] Xueliang Li and Xiaoyan Zhang. The edge split reconstruction problem for chemical trees is NP-complete. *MATCH Communications in Mathematical and in Computer Chemistry*, 51:205–210, 2004.

[17] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, Oxford, 2006.

[18] Robert P. Sheridan and Simon K. Kearsley. Using a genetic algorithm to suggest combinatorial libraries. *Journal of Chemical Information and Computer Sciences*, 35(2):310320, 1995.

[19] Nenad Trinajstić. *Chemical Graph Theory, Second Edition.* CRC Press, 1992.

[20] Harry Wiener. Structural determination of paraffin boiling points. *Journal of the American Chemical Society*, 69(1):17–20, 1947.