Complexity of splits reconstruction for low-degree trees

Serge Gaspers, Mathieu Liedloff, Maya Stein, Karol Suchan

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 e in T is the minimum over the weights of the two trees obtained by removing e from T, where the weight of a tree is the sum of weights of its vertices. Given a set of weighted vertices V and a multiset of integers $\mathcal{S}$, 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 at most 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 at most 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 is a fixed-parameter algorithm for parameter 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 relation between structural invariants of the particles, such as the Wiener index and, less directly, the problem under consideration here, and physico-chemical properties of the substance.

Keywords: reconstruction of trees; computational complexity; computational chemistry

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.

*A preliminary version of this article appeared in the proceedings of WG 2011 [11].

†School of Computer Science and Engineering, UNSW Australia (The University of New South Wales), Sydney NSW 2052, Australia. E-mail: sergeg@cse.unsw.edu.au

‡Optimisation Research Group, NICTA (National ICT Australia), Sydney NSW 2052, Australia

§Université d'Orléans, INSA Centre Val de Loire, LIFO EA 4022, FR-45067 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.

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

> 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 on V whose multiset of splits is $\mathcal{S}$ (that is, $\mathcal{S}(T) = \mathcal{S}$)?

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 subclass 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, 20], 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 [14]): 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 [23] 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 properties 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. [13] 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 [19] and [12]. Goldman et al. show that for every positive integer W, except 2 and 5, there exists a graph with Wiener index W. 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 [23]: 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. Independently, Wagner [21] and Wang et al. [22] show that all but a finite number of integers are Wiener indices of trees.

As it is not reasonable to construct chemical trees with arbitrarily high vertex degrees, Li and Zhang [17] 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 [17], 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 also 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 [16] which aim at identifying parameters for which the problem becomes tractable when these parameters are small.

Recall that a problem is strongly NP-complete if it remains so even when all of its numerical parameters are bounded by a polynomial in the length of the input. 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.

Multivariate complexity theory [5, 7, 9, 18] – also known as parameterized complexity – 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

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

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 $\mathrm{W}[t]$, $t \geq 1$, are parameterized intractability 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 [8], but it remains open whether this problem is fixed-parameter tractable. As a relevant parameter for SR we identified the number k 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 [10]).

> 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 and SR_3 are 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. Section 4 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 5 and we conclude with some directions for future research in Section 6.

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. We may assume that $\sum_{i=1}^{n} j_i = d_{n-1} + d_n$, otherwise we trivially face a No-instance. 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. Note that we obtain $n-1$ splits, all smaller than $\frac{1}{2}\sum_{i=1}^{n} j_i$.

Suppose we obtained a YES-instance of WSR_2 and the path $P = (v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)})$ is a solution. Let us simplify descriptions, wherever appropriate, by assuming that a path is drawn horizontally, with indices ordered from left to right, and that whatever is to the left appears "before" whatever is to the right. Say $\{v_{\pi(\ell)}, v_{\pi(\ell+1)}\}$ is the edge associated to the split d_{n-1}. Then the original instance of SCD also was a YES-instance. Indeed, we can 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 all deadlines except for d_n are represented by splits in our instance of WSR_2. Since splits are smaller than $\frac{1}{2}\sum_{i=1}^{n} j_i$, the ones that appear before $\{v_{\pi(\ell)}, v_{\pi(\ell+1)}\}$ in P correspond to the sums of weights of vertices placed to the left of the corresponding edge, and the ones that appear after it correspond to the sums of vertices placed to the right of the corresponding edge. Thus the jobs assigned to P_1 and P_2 satisfy precisely the corresponding deadlines $d_1, \ldots, d_{n-1}$. Finally, 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.

In the other direction, if we have a YES-instance of SCD, then we obtain a YES-instance of WSR_2 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, the minimum value is maintained as a split). Thus, SCD is polynomial-time-reducible to WSR_2.

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

Notice that the restriction on the reconstructed tree to be a path is fundamental in the proof. If we remove this restriction, it is not difficult to create examples where the many-one reduction does not work, that is, where NO-instances of SCD get transformed into YES-instances of unrestricted WSR with the reconstructed tree not being a path. So the decision problem of SCD cannot be easily reduced to the decision problem of WSR. Nevertheless, maybe SCD could be Cook-reduced to the search version of the WSR problem, transforming YES-instances of SCD to YES-instances of WSR where the reconstruction tree is a path. Such a reduction could be based on the following conjecture.

Conjecture 3. *If an instance of the search version of* WSR *has a solution that is a path, then all solutions to this instance are paths.*

However, in this article we focus on the complexity of decision problems. In the remainder of this section, we show that dNMTS is polynomial-time-reducible to SCD. The dNMTS problem is like 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. [15]. 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. There also will be some auxiliary jobs introduced. By a careful construction of job-lengths we ensure that the solutions of YES-instances of SCD created must have a very particular structure.

A convenient way to represent an element $s \in S$ is by introducing a time segment which is delimited to the left and the right by double deadlines, and whose length is equivalent to s. These double deadlines enforce that there is no proper overlapping between a segment and the jobs. Indeed, by the double deadline at the beginning of the segment, both processors must be assigned jobs that end precisely at this moment. The same holds for the end of a segment. Moreover, the elements of $A \cup B \cup S$ are inflated by well-chosen additive factors that preserve solutions in order to assure that the length of each segment can only be met by the sum of lengths of exactly two jobs corresponding to elements of $A \cup B$, one of A and one of B.

Our reduction will create an instance whose solutions assign, 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 one of the processors (the same one), 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-jobs are scheduled first. As we must not introduce any restriction as to which x-jobs can be assigned to which segments, inside each segment we introduce a deadline for each length of an x-job (the starting time of the segment plus the job-length); these are the *real deadlines*.

The job lengths are inflated in a way that in each segment, exactly one processor starts with an x-job, and in each segment, exactly one processor ends by executing a y-job (there is no other way of organizing them).

We refer to the x- and y-jobs as green jobs. We do not want the green jobs to overlap. This is ensured by modifying all deadlines created so far and the corresponding job lengths by a multiplicative factor of 2, and introducing a *fake deadline* at the odd position just one unit before each real deadline. If an x-job and a y-job overlapped, one on processor P_1 the other on P_2, there would be no job ending at the fake deadline preceding the real deadline at which the x-job ends. Indeed, all green jobs have even length and all real deadlines and double deadlines are even.

Blue, red, and black jobs are auxiliary jobs created to make it possible to meet all deadlines that are not served by executing greed jobs. Inflating of the elements of $A \cup B \cup S$ ensures that the auxiliary jobs cannot equate in length the green jobs (except for the black jobs whose lengths might equal the lengths of green y-jobs, but, without loss of generality, one can assign them to the end parts of segments on processors where no green job is being executed).

Moreover, we set that the sum of lengths of all green jobs equals the sum of lengths of the segments. Since the green jobs cannot overlap, this yields that if all deadlines are met, then no auxiliary job (blue, red or black) is scheduled between two green jobs. Therefore, the placement of green jobs in a satisfying scheduling can be easily translated to a solution of the original dNMTS instance.

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 the elements are listed in increasing orders: $a_i < a_{i+1}$, $b_i < b_{i+1}$, $s_i < s_{i+1}$, for all $i \in \{1, \ldots, m-1\}$; and for the biggest elements we have: $a_m < b_m$ (otherwise we switch the sets A and B) and $s_m \leq a_m + b_m$ (otherwise s_m could not be reached as a sum of elements from A and B and we would have a trivial No-instance).

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:

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

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 $y_1 > x_n$, $z_1 > y_n$, and that $2 \cdot x_n < z_1$, $2 \cdot y_1 > z_n$. 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:

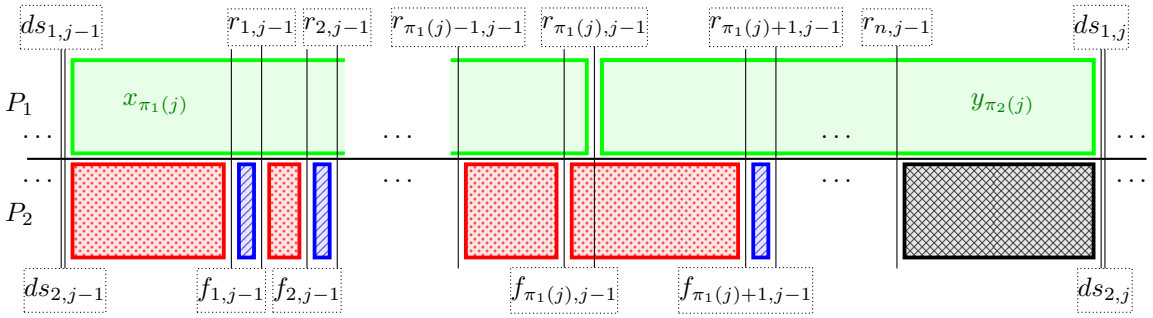

Figure 1: How jobs are assigned to processors in the SCD instance in segment $j < n$.

- *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
- 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 up to z_j.

It is not difficult to check that among different categories of jobs, there are few possible overlaps. A red fill job could have length equal to a blue job (length 1), or a black fill job could have length equal to a green y-job, other classes being disjoint. Moreover, black fill jobs and green y-jobs are all distinct inside their respective classes. So we could have at most 2 jobs longer than 1 of equal lengths, one black fill and one green y-job.

Consider a valid assignment of the jobs to the processors P_1 and P_2. As two jobs with the same length are interchangeable (switching them maintains the same deadlines met), if we find a green y-job in a place where we expect a black fill job, it means that the above mentioned situation occurs and, without loss of generality, we can exchange them. Indeed, according to the construction described below, such a job is needed at most at two places (once as a black fill and once as a green y-job).

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

Proof. Let $j \in \{0, \ldots, n-2\}$. Two jobs must finish at the double deadline $ds_{1,j+1}, ds_{2,j+1}$. One of these must start at $r_{n,j}$ (there are no deadlines in between) 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 (notice there is no black fill job of this length). 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 $ds_{1,n}$, and one processor already satisfies it by Claim2, the other processor needs to process a job finishing at $ds_{1,n}$ and starting before $r_{n,n-1}$. This has to be 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 $y_1, \ldots, y_{n-1}$.

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

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 $ds_{1,p-1} + y_\ell > ds_{1,p-1} + x_n$, by Properties 2 and 3, and the deadline following $r_{n,p-1} = ds_{1,p-1} + x_n$ is $ds_{1,p}$, so it must be that the green y-job y_ℓ finishes at $ds_{1,p}$. Moreover, $ds_{1,p} - y_\ell$ is equal to a real deadline as $ds_{1,p} - y_\ell$ is even. □

Each of the $2n$ jobs that have been assigned so far finishes 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 $0 \le j \le n-1$.*

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 $[ds_{1,j}, f_{1,j}]$. 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 $x_\ell > r_{n,p-1} - f_{1,p-1}$. Indeed $r_{n,p-1} - f_{1,p-1} = x_n - x_1 + 1$ and, by construction, $x_n - x_1 + 1 \leq 2b_m$, whereas $x_\ell \geq 2b_m + 4$. Moreover, $r_{n,p-1}$ is the latest deadline strictly inside the segment p. So the green x-job x_ℓ has to start at $ds_{1,p-1}$. Notice that $ds_{1,p-1} + x_\ell < ds_{1,p}$ and that $ds_{1,p-1} + x_\ell$ corresponds to a real deadline as $ds_{1,p-1} + x_\ell$ 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 (that is, 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. □

By Claims 1, 2, 3, 4, 5, 6 there is exactly one green x-job and one y-job assign to each segment. By Claim 7, these jobs do not overlap. Since the sum of lengths of green jobs is equal the sum of lengths of segments, this 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 segment. 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 4. *dNMTS* $\leq_p$ SCD.

We have assembled enough information to prove our main theorem.

Theorem 5. WSR_2 *is strongly* NP*-complete.*

Proof. The theorem follows from the strong NP-hardness of dNMTS, Lemmas 2 and 4, 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. □

With this result, we can show that Splits Reconstruction with unit weights is NP-complete for trees with maximum degree 3.

Theorem 6. SR_3 *is* NP*-complete.*

Proof. It is clear that this problem 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 Ωn (so, for $1 \leq i \leq n-3$, $\omega_i = \omega'_i \Omega n$, and $s_i = (s'_i + \Omega)\Omega n$).

It is not difficult to see that I_P and I'_P are equivalent. Indeed, the two additional vertices of weight $n\Omega^2$ are the heaviest vertices in the new instance and the two additional splits of the same value are the smallest splits. Therefore, any solution has to put these two as the end vertices of the path. Given this "border condition", the rest of the instance is clearly equivalent to the original one, just with all values multiplied by a constant.

Now let us create an instance I_C of SR_3 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 in I_C there are $\sum_{i=1}^{n} \omega_i$ vertices and $(\sum_{i=1}^{n} \omega_i) - 1$ splits (that is, edges) in total.

If I_P is a YES-instance then I_C is a YES-instance. Indeed, let $P = (v_1, v_2, \ldots, v_n)$ be a solution to I_P, with weights and splits assigned accordingly. We create a solution to I_C by keeping the path P with the splits associated to its edges as the *backbone*, and applying the following transformation. For each vertex v_i on the backbone, create a path of $\omega(v_i) - 1$ copies of v_i and attach it as a *hair* to the original v_i. To each vertex assign a weight 1. To each edge e on a hair, assign a split of value equal to the number of vertices on the path obtained by removing e (disconnected from the caterpillar). Keep the splits on the backbone. It is not difficult to check that the graph we obtain is a solution to I_C.

Now suppose I_C is a YES-instance with a solution C. To analyze the structure of C, organize the splits in I_C in ascending order and observe to which edges they are associated.

By construction of I_C, there are n splits of value 1, and each of them has to be associated to an edge incident to a leaf, an end vertex of a hair of C. Notice that there are no other leaves in C. Contract each of these edges and assign the resulting vertex the weight 2. It is a special case of assigning the new vertex the sum of weights of the original vertices. This operation preserves the correctness of splits association, only that we get to work with a weighted graph again. Let us denote it by C'.

Now proceed to splits of value 2, there is also n of them. They have to be assigned to edges incident to leaves in C'. Like in the first step, contract these edges and assign the new vertices the sum of weights of the original ones. We proceed in this way with all edges that are assigned splits of value strictly smaller than $\Omega^2 n$. Let C'' denote the graph that we obtain in the end. It is not difficult to check that C'' has $2n$ vertices, n leaves and $2n - 1$ edges. The splits assigned to them are the ones obtained from original splits: $s_i = (s'_i + \Omega)\Omega n$, for $1 \leq i \leq n - 3$, two added ones of value $\Omega^2 n$, the auxiliary ones that correspond to original vertices: $\omega'_i \Omega n - 1$, for $1 \leq i \leq n - 2$, and the two auxiliary ones of value $\Omega^2 n - 1$ that correspond to the added vertices. Notice that the auxiliary splits are assigned to the edges incident to the leaves of C''.

Let us pick e, one of the edges assigned a split value of $\Omega^2 n$, remove it from C'' and analyze the structure of the connected component D of total weight smaller than $\Omega^2 n$. Let v denote the vertex of D that is incident to e in C''. Let us analyze the edges of D incident to v. Since C'' is of maximum degree 3, there are at most 2 of them. In fact, let us show that it is only one, and thus its assigned split is $\Omega^2 n - 1$. On the contrary, suppose there were two different edges f, g incident to v in D. Then there would be $s(f) + s(g) = \Omega^2 n - 1$, with $s(f) = \omega'_k \Omega n - 1$ and $s(g) = \omega'_l \Omega n - 1$, for some $1 \leq k, l \leq n - 2$. But it is impossible, since $\Omega := 1 + 2 \cdot \max\{\omega'_i : 1 \leq i \leq n - 2\}$. Notice that it means that D is just a path of length 1, and the same has to be true about the other edge assigned a split of value $\Omega^2 n$.

Consider the other edges that are assigned auxiliary splits. Since C'' is connected and its maximum degree is at most 3, there cannot be three of them incident to the same vertex. In fact, each of them has to be incident to a unique non-leaf vertex. Indeed, suppose we had two such edges a, b incident to the vertex w. Let c be the third edge incident to w, it has to be assigned a split created from an original one (of kind $s(c) = (s'_i + \Omega)\Omega n$, for some $1 \leq i \leq n - 3$). By correctness of splits assignment, there has to be $s(a) + s(b) + 1 = s(c)$. But it is not difficult to check that it is not possible, by construction of I_C.

To sum up, we have shown that each of the n leaves of C'' is adjacent to a unique non-leaf vertex. Therefore, given that C'' is a tree of maximum degree 3 on $2n$ vertices, the only possible topology of C'' is that of a caterpillar with n leaves and n vertices on the backbone B. It is not difficult to see that the splits assigned to the edges on the backbone give us a solution to I_C. □

Notice that the solution C to I_C constructed in the proof above has to be a caterpillar of maximum degree 3 with hair of unbounded length. Therefore, we have also shown the following corollary.

Corollary 7. SPLITS RECONSTRUCTION FOR CATERPILLARS OF UNBOUNDED HAIR-LENGTH AND MAXIMUM DEGREE 3 *is* NP*-complete.*

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, that is $m_i = |\{v \in V \,:\, \omega(v) = w_i\}|$ 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 $A[p, W_L, W_R, n_1, n_2, \ldots, n_k]$ 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 $A[p, W_L, W_R, n_1, n_2, \ldots, n_k]$ is set to `true` if and only if there is an assignment of the first p 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 the 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. Notice that given a partial solution for the first p splits that corresponds to a postive entry $A[p, W_L, W_R, n_1, n_2, \ldots, n_k]$, the knowledge of exact values of ℓ and r, and the precise assignment of weights to vertices on the left and the right side of P_n is not necessary in order to know if this partial solution can be extended to a complete one. The information conveyed in the values of indices of the table is sufficient. Actually, there even is some redundancy, given that the value of p can be computed based on the values of n_i, $1 \leq i \leq k$.

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.

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

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 8. 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, observe that $A[p, W_L, W_R, n_1, n_2, \ldots, n_k]$ is assign `true` if and only if the first $p-1$ splits can be assigned to the edges of P_n in a way that can be extended to a partial solution satisfying the conditions described by the indices. More specifically, it means that there is a weight ω_i available (notice the increase of the number n_i of weights ω_i used indicated in the formulas), for some $1 \leq i \leq k$, that can be assigned to a vertex that extends the left (first part of the internal

alternative) or the right part, respectively, of a previously checked partial solution and so the new split, that is the frontier between the assigned and unassigned part of the path, is equal $W_L = (W_L - \omega_i) + \omega_i$ or $W_R = \omega_i + (W_R - \omega_i)$, respectively. Clearly, this is the only way possible of obtaining a partial solution for the first p splits with the corresponding multi-set of weights used.

Finally, a complete assignment is found if there exists a weight ω_i and splits W_L and W_R, such that there exists a partial solution using all weights but one instance of ω_i, the left part ends with the split W_L, the right part ends with W_R, and these values satisfy the condition on the split assigned to an edge ℓ to be the smaller of the weights of the components obtained by removing ℓ from the graph.

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 Algorithm for SR with few leaves

Recall that an instance of the SR problem is given by a set $\mathcal{S}$ of splits. The number of vertices in the tree that we are looking for is equal to $|\mathcal{S}| + 1$ (with the multiplicities taken into account). We will assume that $|\mathcal{S}| > 1$. In this section we design an algorithm for SR parameterized by the number k of splits that are equal to one, that is $k = |\{s = 1 : s \in \mathcal{S}\}|$. As such splits are exactly the ones that correspond to an edge incident to a leaf in the reconstructed tree, the algorithm reconstructs trees with k leaves.

Before we go into details, let us first make some observations.

Observation 9. *Let T be a tree with a valid assignment of splits. There can be no split of value x at most b assigned to an interior edge e on a path connecting two edges f, g assigned splits of value at least b.*

Indeed, suppose there are such edges e, f, g. The split x assigned to the edge e means that T_1, one of the two components obtained by removing e from T, has total weight x. Without loss of generality, suppose that T_1 contains the edge f. Then, the total weight of the component obtained by removing f from T that does not contain e is strictly smaller than b. But this contradicts the fact that the split assigned to f is larger than b.

Observation 10. *Let T be a tree with a valid assignment of splits. Then there exists a vertex r in T such that, for any leaf f, the splits along the path from r to f are strictly decreasing.*

In order to see this fact, consider the edges assigned the maximum split. If there are at least two such edges, they all must share one unique vertex r (as by Observation 9 any two such edges are adjacent). In this situation, for any leaf f of T the split assigned to an edge e on the path from r to f is the weight of the component obtained by removing e from T that contains f. Since the weights of these components strictly decrease as we get closer to f, the observation follows. Now, if there is only one edge of maximum split value, it is not difficult to check that the same statement is true for at least one of the incident vertices.

Our algorithm is based on the existence of such a vertex r, as described in Observation 10. Such a vertex is considered the root of T. We start with a very rough approximation of T that consists just of the root r_0 and k leaves, organized as a star. The leaves are assigned unit weight, the incident edges have unit splits. The root is assigned the weight $\omega(r_0) = n - k$. The leaves will stay unchanged until the end of the algorithm execution, but the paths from the root will get progressively improved to finally reach the form they have in the tree T that we are looking for. In each step of the algorithm we will improve upon the paths that have the lowest value of the split incident to the root.

So, initially we have the star T_0. To update the information on available (not yet assigned) splits, we define $\mathcal{S}_0$ to be $\mathcal{S}$ minus the unit splits already assigned to the star. Notice that this way there are no splits of value 1 in $\mathcal{S}_0$.

At each step i of the algorithm, we construct a new tree T_i by applying a particular kind of transformation to T_{i-1} and choosing a new root r_i. Throughout the algorithm, r_i is the only vertex allowed to have a non-unit weight. Initially, T_{i-1} is a tree with the splits from $\mathcal{S} \setminus \mathcal{S}_{i-1}$ assigned to its edges. The root r_{i-1} is adjacent to the vertices in its *neighborhood* $N_{i-1} := N(r_{i-1})$.

The goal is to find a tree R_i with the root r_i and the set of leaves L_i, that can be considered a kind of subdivision of r_{i-1}, and construct a new tree T_i by replacing r_{i-1} in T_{i-1} with R_i. The total weight of R_i has to be equal to $\omega(r_{i-1})$ and distributed between $\omega(r_i)$ and unit weights of other vertices of R_i.

The tree R_i may have some other vertices besides the root and the leaves. In the construction, r_{i-1} is removed and each node of L_i is made adjacent to one or more vertices in N_{i-1}, while each vertex of N_{i-1} is made adjacent to exactly one node of L_i, and the tree T_i is obtained. The splits assigned to the new edges have to be some values chosen from $\mathcal{S}_{i-1}$, the other ones being put in $\mathcal{S}_i$.

If there exists such a transformation where r_i is subdivided into a tree with unit weights on all vertices, we say that T_i has a *valid extension*. Let us also extend the standard notion of *parent-child* relation between nodes in a rooted tree to analogous relation between edges. Say that edge e is the parent of f if they share a vertex v and the path from r to v contains e.

Using this term and based on Observation 10 we can say that our algorithm will iteratively choose a set of edges F incident to the root r, with the corresponding set N of vertices adjacent to r; "subdivide" r by creating its "clone" r' that is incident to the edges in F, whereas r keeps the incidence to other edges; and a new edge rr' is created, parent to all edges in F and assigned a split of value equal to the sum of splits assigned to edges in F. It is easy to check that by inverting this process, starting from a solution T and iteratively contracting edges incident to the root and not to a leaf, we eventually get T_0. Let us now get into the details.

The tree T_{i-1} that we have at the beginning of step i uniquely defines a partition (A, C, U) of the splits $\mathcal{S}$ such that

- A represents the multiset of *available* splits that have not been assigned to T_{i-1} (stored as $\mathcal{S}_{i-1}$),
- C represents the multiset of *current* splits assigned to edges incident to r_{i-1}, and
- U represents the multiset of *used* splits assigned to edges of T_{i-1} that are not incident to r_{i-1}.

Let b denote the value of the smallest split in C. Our tree T_{i-1} will *grow out* of r_{i-1} as follows.

- If $\omega(r_{i-1}) = 1$, then return TRUE. Indeed, since all vertices are now assigned unit weights, there are $|\mathcal{S}| + 1$ vertices and $|\mathcal{S}|$ splits assigned to edges in T_i. Thus we have a solution.
- If A contains a split whose value is at most b, then by Observation 9, we know that T has no valid extension and the algorithm backtracks.
- If $a := |\{s \in A : s = b + 1\}| > c := |\{s \in C : s = b\}|$, that is, A contains strictly more splits with value $b + 1$ than C contains splits with value b, then T has no valid extension and the algorithm backtracks. Indeed, by Observation 9, there can be no split of value $b + 1$ assigned to an edge on a path connecting two edges of split value $b + 1$. So all a splits in A with value $b + 1$ would have to be assigned to new edges which are parents of edges in C of split value b. Which is impossible, since $a > c$.
- If $|\{s \in A : s = b+1\}| = |\{s \in C : s = b\}|$, then all valid extensions of T_{i-1} are also valid extensions of the tree obtained from T_{i-1} by subdividing each edge with split b that is incident to r_{i-1}. That is, for each edge $r_{i-1}v$ with a split of value b, add a new vertex z_v, remove the edge $r_{i-1}v$, and add edges $r_{i-1}z_v$ and $z_v v$ with splits assigned accordingly. To finally obtain T_i, replace r_{i-1} with r_i, with assigned weight $\omega(r_{i-1}) - |\{s \in C : s = b\}|$. Define $\mathcal{S}_i$ as $\mathcal{S}_{i-1} \setminus \{s \in A : s = b + 1\}$. The algorithm proceeds recursively on T_i.
  Note that by Observation 9, and an argument similar to the one in the previous case, any valid extension of T_{i-1} is also a valid extension of T_i.
- Otherwise, that is if $|\{s \in A : s = b + 1\}| < |\{s \in C : s = b\}|$, we need to create a branch where an edge assigned split b receives a parent edge with split value more than $b+1$. Go over all choices for selecting a subset U of $N(r_{i-1})$ of size at least 2 containing a vertex v such that $r_{i-1}v$ is associated with a split with value b.

  If A contains no split that equals $1 + \sum_{u \in U} s(r_{i-1}u)$, then discard this choice. Otherwise, create a new vertex z_U, remove the edges $\{r_{i-1}u : u \in U\}$ from T_{i-1}, add the edges $\{z_U u : u \in U \cup \{r_{i-1}\}\}$, and replace r_{i-1} with r_i of weight equal $\omega(r_{i-1}) - 1$ to obtain T_i^U. Assign the splits correspondingly and set $\mathcal{S}_i^U = \mathcal{S}_{i-1} \setminus \{1 + \sum_{u \in U} s(r_{i-1}u)\}$.

  The algorithm recursively solves each of the resulting subproblems. The tree T_{i-1} has a valid extension if and only if one of the trees T_i^U has a valid extension.

Theorem 11. 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 the maximum number of atomic instances solved for an instance with $|C| \leq c$, 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 $T(c)$ is non-decreasing, and $\binom{c}{i} \leq c^i$, we have that

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

The last inequality holds, since the recurrence $T(n) = a \cdot T(n-b)$, with $a, b \in O(1)$ and $T(d) = O(1)$ for $d = O(1)$, solves to $T(n) = O(a^{n/b})$. This maximum is attained for $i = 2$, and the theorem now follows since $c \leq k$. □

5 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 12. 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 $K_{1,\kappa}$, $m(j) - 1$ 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 12, 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, any given set of splits can be realized in the same way as explained in the proof of Theorem 12 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 adjacent to v. Thus, if $\mathcal{S}$ contains a sufficient number of splits 1, then $\mathcal{S}$ can 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

$$\kappa + \sum_{j=1}^{\kappa} \sum_{i=1}^{m(j)} (s_i^j - s_{i-1}^j - 1) + \max \left(0, 2 \cdot \max_{1 \leq i \leq \kappa} \{ s_{m(i)}^i \} - 1 - \sum_{j=1}^{\kappa} s_{m(j)}^j \right)$$

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 $\max\left(0, 2 \cdot \max_{i=1}^{\kappa}\{s_{m(i)}^i\} - 1 - \sum_{j=1}^{\kappa} s_{m(j)}^j\right)$ of them are added, with pending vertices, to the root. The latter term ensures that whenever an edge corresponding to a split $s_{m(j)}^j$ is removed the weight of the connected component corresponding to the subdivision of e_j is at least the weight of the other connected component. (Note that it does not matter if there are more splits 1 than needed in our construction, since we may always add leaves adjacent to the root 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.

6 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 [8]). 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.

Acknowledgments

We thank Ming-Yang Kao for communicating this problem.

All authors acknowledge the support of Conicyt Chile via projects Fondecyt 11090390 (Mathieu Liedloff, Karol Suchan), Fondecyt 11090141 (Maya Stein), and Basal-CMM (Serge Gaspers, Maya Stein, Karol Suchan). Serge Gaspers is the recipient of an Australian Research Council Discovery Early Career Researcher Award (project number DE120101761). NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. Mathieu Liedloff and Karol Suchan acknowledge the support of the French Agence Nationale de la Recherche (ANR AGAPE ANR-09-BLAN-0159-03).

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 Rosamond. Multivariate complexity theory. In Edward K. Blum and Alfred V. Aho, editors, *Computer Science: The Hardware, Software and Heart of It*, chapter 13, pages 269–293. Springer, 2011.

[8] Michael R. Fellows, Serge Gaspers, and Frances A. Rosamond. Parameterizing by the number of numbers. *Theory of Computing Systems*, 50(4):675–693, 2012.

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

[10] 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.

[11] Serge Gaspers, Mathieu Liedloff, Maya Stein, and Karol Suchan. Complexity of splits reconstruction for low-degree trees. In Petr Kolman and Jan Kratochvíl, editors, *Graph-Theoretic Concepts in Computer Science*, volume 6986 of *Lecture Notes in Computer Science*, pages 167–178. Springer Berlin Heidelberg, 2011.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

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

[21] Stephan G. Wagner. A class of trees and its Wiener index. *Acta Applicandae Mathematica*, 91(2):119–132, 2006.

[22] Hua Wang and Guang Yu. All but 49 numbers are Wiener indices of trees. *Acta Applicandae Mathematica*, 92(1):15–20, 2006.

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