Refining the complexity of the sports elimination problem

Katarína Cechlárová[1]*, Eva Potpinková[1]*, and Ildikó Schlotter[2]**

[1] Institute of Mathematics, Faculty of Science, P.J. Šafárik University
Jesenná 5, 040 01 Košice, Slovakia
katarina.cechlarova@upjs.sk,eva.potpinkova@student.upjs.sk
[2] Budapest University of Technology and Economics
H-1521 Budapest, Hungary
ildi@cs.bme.hu

Abstract. The sports elimination problem asks whether a team participating in a competition still has a chance to win, given the current standings and the remaining matches to be played among the teams. This problem can be viewed as a graph labelling problem, where arcs receive labels that contribute to the score of both endpoints of the arc, and the aim is to label the arcs in a way that each vertex obtains a score not exceeding its capacity. We investigate the complexity of this problem in detail, using a multivariate approach to examine how various parameters of the input graph (such as the maximum degree, the feedback vertex/edge number, and different width parameters) influence the computational tractability. We obtain several efficient algorithms, as well as certain hardness results.

Keywords: sports elimination problem; graph labelling; parameterized complexity; multivariate complexity analysis.

1 Introduction

1.1 Motivation

Imagine we are in the middle of an ice-hockey[3] season. Each participating team has currently a certain score and still some matches to play. Can our favorite team t_0 become a winner of the season? More precisely, given the current scores and the set of remaining matches, is it possible that these matches end in such a way that our team will finish with the maximum score among all teams? If the answer is not, our team is said to be *eliminated*. This is a question that occupies not only players, coaches and managers of teams, but also many sports fans. It has also attracted quite a lot of attention from mathematicians and computer scientists. Papers [1] and [22] use integer linear programming to solve

* Supported by the grants VEGA 1/0344/14, 1/0479/12.
** Supported by the Hungarian Scientific Research Fund (OTKA grants no. 108383 and no. 108947).
[3] The reader may substitute any game he or she likes.

this problem, but we shall concentrate more on combinatorial approaches, see [4, 10–12, 15, 16, 20, 23, 24].

1.2 Formulation of the problem

Let us suppose that the rules of the game define the set of outcomes of a match as

$$S = \{(\alpha_0, \beta_0), (\alpha_1, \beta_1), \ldots, (\alpha_k, \beta_k)\}.$$

This formalism corresponds to situations where each match has a 'home' team and an 'away' team, and it can end in any of the $k+1$ ways with the home team getting α_i points and the away team β_i points. For example, $S = \{(0,1), (1,0)\}$ for baseball, as this game does not allow draws, and a winning team gets 1 point. Basketball, where the winning team gets 2 points, and both teams in a match that ends in a draw are awarded 1 point, has $S = \{(0,2), (1,1), (2,0)\}$. European football differs from basketball in that the winner gets 3 points, so $S = \{(0,3), (1,1), (3,0)\}$ for European football. Examples of other games are given by Kern and Paulusma [16].

A polynomial-time reduction [4] provided also by the same authors [16] showed that we can restrict ourselves to the case where

$$\alpha_0 = 0,\ \alpha_1 = 1 < \alpha_2 < \cdots < \alpha_k \text{ and } \beta_0 > \beta_1 > \cdots > \beta_{k-1} \geq 1,\ \beta_k = 0. \quad (1)$$

The set of outcomes fulfilling (1) is called *normalized*. Throughout the paper we will assume S to be normalized.

An instance of the Generalized Sports Elimination problem with the set S of outcomes (GSE(S) for short) can be described by a triple $(\mathcal{T}, w, \mathcal{M})$. We let $\mathcal{T} = \{t_0, t_1, \ldots, t_n\}$ represent the set of teams participating in the competition. The function $w : \mathcal{T} \to \mathbb{R}$ defines current scores and $\mathcal{M} : \mathcal{T} \times \mathcal{T} \to \mathbb{N}$ the number of remaining matches between teams of $\mathcal{T}$.

By a (t, t')-match for some $t, t' \in \mathcal{T}$ we mean a match played between t and t' such that t is the home team and t' is the away team. The question is whether it is possible that all the remaining matches end in such a way that team t_0 will have the maximum score among all teams. More precisely, given the set of outcomes S, the problem GSE(S) is defined as follows.

Generalized Sports Elimination for S:
Instance: A triple $(\mathcal{T}, w, \mathcal{M})$ as described above.
Question: Can a final score vector $s : \mathcal{T} \to \mathbb{R}$ be reached such that $s(t_0) \geq s(t_i)$ for each $t_i \in \mathcal{T}$?

If the answer is yes, we say that team t_0 is *not eliminated*, otherwise t_0 is *eliminated*. Observe that we can suppose that our team t_0 has already played all its matches, and in each one it obtained the maximum possible score, so its final

[4] The reduction does not change the directed graph underlying the instance (formally defined later on), except for possibly reversing all its arcs.

standing is $w(t_0)$ points. (If in reality this is not the case, we can modify the values of w accordingly).

An instance $(\mathcal{T}, w, \mathcal{M})$ of GSE(S) can quite naturally be represented by a directed multigraph $G = (V, A)$ with vertex capacities $c : V \to \mathbb{R}$. The vertex set $V = \{v_1, \dots, v_n\}$ of G corresponds to the teams $\mathcal{T} \setminus \{t_0\}$, and arcs $(v_i, v_j) \in A$ correspond to the remaining matches between teams t_i and t_j. More precisely, the multiplicity of an arc (v_i, v_j) equals the number of remaining (t_i, t_j)-matches. The capacity of a vertex $v_i \in V$ is equal to $c(v_i) = w(t_0) - w(t_i)$, and it represents the number of points that team t_i can still win so as not to overcome team t_0. It is easy to see that GSE(S) is equivalent to the following problem that we call ARC LABELLING WITH CAPACITIES FOR S, or ALC(S) for short.

Arc Labelling with Capacities for $S = \{(\alpha_0, \beta_0), \dots, (\alpha_k, \beta_k)\}$:
Instance: A pair (G, c) where $G = (V, A)$ is a directed multigraph and $c : V \to \mathbb{R}$ is a vertex capacity function.
Question: Does there exist an assignment $p : A \to \{0, \dots, k\}$ such that

$$\mathrm{scr}_p(v) := \sum_{a=(v,u)\in A} \alpha_{p(a)} + \sum_{a=(u,v)\in A} \beta_{p(a)} \leq c(v) \qquad (2)$$

holds for each vertex $v \in V$?

We say that $p : A \to \{0, \dots, k\}$ is a *score assignment* for G. If $p(a) = q$ for some arc $a = (u, v) \in A$, then we also say that p assigns the outcome (α_q, β_q) to the arc a, and that u and v *gain* α_q and β_q (resulting) from the arc a, respectively. To keep the notation simple, instead of $p((u, v))$ we shall simply write $p(uv)$. The *score* of some vertex v in p, denoted by $\mathrm{scr}_p(v)$, is defined by the left-hand side of Inequality (2); clearly, $\mathrm{scr}_p(v)$ equals the total points that v gains when all remaining matches yield the outcome as determined by p. We say that a score assignment p for G is *valid* with respect to a capacity function c, if $\mathrm{scr}_p(v) \leq c(v)$ for each vertex $v \in V$. Thus, the task in the ALC(S) problem is to decide whether a valid score assignment exists.

Problem ALC(S) restricted to instances with graphs G having maximum degree at most Δ will be denoted by Δ-ALC(S).

As the reader can see from the definitions of the problems GSE(S) and ALC(S), we take the view that the game (in fact, the set of outcomes S) is fixed, and a different S defines another variant of the elimination problem or of the corresponding graph labelling problem. As a consequence of this assumption, the size of the set S is a constant. However, to guarantee a greater insight into the complexity of the algorithms proposed, we sometimes state running times with their dependence on k made explicit; in all cases where the dependence on k is not explicit, we assume k to be a fixed constant.

1.3 Previous work

If the rules of the game are such that the winner of a match gets 1 point, the loser gets 0 points and there are no draws (like in baseball), that is, $S = \{(0, 1), (1, 0)\}$, then the elimination problem can easily be solved by employing network flow theory. Schwartz [23] was the first one to propose such a method; his network has $O(n^2)$ vertices, where n is the number of teams. Another construction with a network containing only $O(n)$ vertices was proposed by Gusfield and Martel [10].

However, it soon turned out that some score allocation rules make the elimination problem intractable. Bernholt et al. [4] proved that GSE(S) is NP-complete for the European football system where $S = \{(0,3),(1,1),(3,0)\}$. They also mentioned that this result could be generalized to the rules that award α points to the winner and β points to both teams of a match ending in a draw, if $\alpha > 2\beta$. Kern and Paulusma [15, 16] extended this result by classifying all score allocation rules S into polynomially solvable and NP-complete cases. Their results show that GSE(S) is NP-complete for any possible (normalized) set S of outcomes, except for the case when $S = \{(i, k-i) \mid 0 \leq i \leq k\}$ for some $k \in \mathbb{N}$, which makes the problem polynomial-time solvable.

In their NP-completeness proofs, Kern and Paulusma [15, 16] gave reductions from the 3-dimensional matching problem (3DM for brief). It is known that the restriction of 3DM to instances where each element occurs in at most three triples remains NP-complete (see [9], problem SP1). Applying the reduction given by Kern and Paulusma [16] to such instances, one obtains that the following stronger result holds as well:

Theorem 1 *Except when the set of outcomes is of the form $S = \{(i, k-i) \mid 0 \leq i \leq k\}$ for some $k \in \mathbb{N}$,* GSE(S) *is NP-complete even in the case when no team has more than 3 remaining matches to play.*

Considering Arc Labelling with Capacities, Theorem 1 immediately yields the following assertion.

Corollary 1 3-ALC(S) *is NP-complete unless $S = \{(i, k-i) \mid 0 \leq i \leq k\}$ for some $k \in \mathbb{N}$.*

By contrast, Bernholt et al. [4] noticed that the European football elimination problem can be solved in polynomial time if there are at most two remaining matches for each team; this means that 2-ALC($\{(0,3),(1,1),(3,0)\}$) is polynomial-time solvable.

We remark that although Arc Labelling with Capacities seems to be a very natural variant within the broad class of graph labelling problems, the authors are not aware of any prior work studying this problem explicitly, apart from the research focusing on sports elimination which deals with it implicitly.

1.4 Our contribution

We investigate the computational aspects of the elimination problem by studying the complexity of Arc Labelling with Capacities in detail. Instead of focusing only on how the set S of outcomes determines the complexity of the problem, we look more closely on how the structure of the input graph alters its tractability.

We first identify certain conditions that render our problem polynomially solvable. Namely, we show tractability of the case when the undirected version of the input graph is a forest, by proposing a simple linear-time dynamic programming algorithm in Theorem 2. We also assert polynomial-time solvability of the cases when the undirected version of the input graph is a cycle (Theorem 3) or a graph with maximum degree 2 (Theorem 4). The latter theorem, stating that 2-$\textsc{alc}(S)$ is in P for every S, extends the result of Bernholt et al. [4] (dealing with the European football elimination problem) and is in sharp contrast to the fact that 3-$\textsc{alc}(S)$ remains NP-complete.

After identifying easy cases of $\textsc{alc}(S)$, we perform a thorough investigation on how certain parameters of the input graph modify the tractability of the problem. We apply the framework of parameterized complexity to study how a small deviation from the tractable cases can be handled. To this end, we consider various parameters of the input graph which aim to measure its distance from a certain class of tractable instances. Namely, we focus on three types of parameters of the input graph:

- *degree parameters* such as the number of vertices with degree more than two;
- *feedback parameters*, describing how many vertices or edges must be deleted from the (undirected version of the) input graph in order to obtain a forest;
- *width parameters* such as treewidth and pathwidth, which measure the 'tree-likeness' of the input graph in a more elaborate way.

In the case of degree parameters, we were able to provide efficient fixed-parameter tractable algorithms (Theorems 5 and 6). Interestingly, the tractability of $\textsc{alc}(S)$ with respect to feedback parameters differs greatly depending on whether we deal with the size of a minimum feedback *edge* set (in which case Theorem 7 provides fixed-parameter tractability), or with the size of a minimum feedback *vertex* set (which yields intractability by Theorem 8). Regarding width parameters, in Theorem 9 we obtain that for each fixed integer w, $\textsc{alc}(S)$ becomes polynomial-time solvable on graphs with treewidth at most w. However, this result cannot be strengthened to achieve fixed-parameter tractability, since Theorem 8 also implies W[1]-hardness with respect to the parameter w (using that the treewidth of a graph is at most 1 plus the size of its minimum feedback vertex set). Furthermore, $\textsc{alc}(S)$ turns out to be W[1]-hard when parameterized by the pathwidth of the underlying graph, as proven in Theorem 10. See Table 1 in Section 7 for a summary of the obtained results.

These results provide a detailed insight into the computational complexity of the Arc Labelling with Capacities problem, using the tools of parameterized complexity. The performed analysis follows the methodology proposed by Niedermeier [21], see also the papers [18, 5], investigating several different parameters of the problem to obtain a refined, multidimensional view of its complexity.

1.5 Organization of the paper

We introduce the necessary notation in Section 2. Polynomial-time algorithms for $\textsc{alc}(S)$ are presented in Section 3. Results in connection to degree, feedback and width parameters are contained in Sections 4, 5, and 6, respectively. Finally, we summarize the paper and propose some open questions in Section 7.

2 Notation

2.1 Graph theory

Let us now define some standard notions for undirected graphs. If G is an undirected graph, then $V(G)$ and $E(G)$ denote its set of vertices and edges, respectively; $E(G)$ may contain parallel edges as well as loops. We let $N(v)$ denote the set of *neighbors* of a vertex v, and for some set X of vertices, $N(X) = \bigcup_{v \in X} N(v)) \setminus X$. The *degree* of v is $|N(v)|$. A *path* of length m in G is a sequence $(v_0, v_1, \ldots, v_m)$ of mutually distinct vertices such that $\{v_i, v_{i+1}\} \in E(G)$ for each $i = 0, 1, \ldots, m-1$. If v_0 and v_m are also connected by an edge, then we call $(v_0, v_1, \ldots, v_m)$ a *cycle* of length $m+1$. We say that G is *connected* if any two vertices $u, v \in V(G)$ are joined by a path. The *connected components* of G are its inclusion-wise maximal connected subgraphs.

We say that G is a *forest* if it contains no cycles. A connected forest is a *tree*. The degree-1 vertices in a tree are called *leaves*. A *star* on n vertices is a tree having a vertex v of degree $n-1$ and $n-1$ vertices of degree 1; the vertex v is called the *central vertex* of the star.

If X is a set of vertices or edges in G, then $G - X$ is the graph obtained by deleting the elements of X from G. When deleting vertices, we also delete all incident edges. We may write $G - x$ instead of $G - \{x\}$. We also let $G[X] = G - (V(G) \setminus X)$ denote the subgraph *induced* by X. For a graph G, we let G^1 denote the simple version of G, obtained by deleting loops and replacing each set of parallel edges by a single edge in G.

A *feedback vertex set* of G is a set $U \subseteq V(G)$ such that $G - U$ is a forest. Similarly, a *feedback edge set* of G is a set $U \subseteq E(G)$ such that $G - U$ is a forest. The minimum size of a feedback vertex set of G is its feedback vertex set number, denoted by $\mathrm{fvs}(G)$; the feedback edge set number $\mathrm{fes}(G)$ is defined analogously.

Let us now turn to directed graphs. Given a directed graph G, we denote its vertex set by $V(G)$ and its multiset of arcs by $A(G)$. The number of arcs *leaving* a vertex $v \in V$ is the *out-degree* of v, denoted by $\delta^+(v)$, the number of arcs *entering* v is its *in-degree* and is denoted by $\delta^-(v)$. The *degree* of v is the sum $\delta^+(v) + \delta^-(v)$. Some vertex v is an *in-neighbor* of a vertex u if $(v, u) \in A(G)$, and v is an *out-neighbor* of u if $(u, v) \in A(G)$. A *directed path* in G is a sequence $(v_0, v_1, \ldots, v_m)$ of mutually distinct vertices such that (v_i, v_{i+1}) is an arc of G for each $i = 0, 1, \ldots, m-1$.

We denote the undirected graph underlying G by $\bar{G}$. We say that G is *connected* if and only if $\bar{G}$ is connected; the connected components of G are again its inclusion-wise maximal connected subgraphs. A directed graph G is called an *out-tree* with *root* $r \in V(G)$ if $\bar{G}$ is a tree, r has in-degree 0 and every other vertex has in-degree 1 in G.

Notations $G - X$, $G[X]$ and G^1 for directed graphs are defined analogously as in the undirected case.

A *tree-decomposition* of an undirected graph G is a pair $\mathbb{T} = (T, (B_t)_{t \in V(T)})$ where T is a tree, and $B_t \subseteq V(G)$ for each $t \in V(T)$ such that the following conditions hold:

– for any $v \in V(G)$, there exists some node t with $v \in B_t$,
– for any $\{u, v\} \in E(G)$, there exists some node t with $u, v \in B_t$,
– for any $v \in V(G)$, the nodes $\{t \mid v \in B_t\}$ form a connected subtree of T.

The *width* of a tree-decomposition is the maximum of $|B_t| - 1$ taken over all $t \in V(T)$. The *treewidth* of G is the minimum width of any possible tree-decomposition of G.

A tree-decomposition is *nice*, if T is a rooted binary tree, and each node $t \in V(T)$ is one of the following types:

– a *leaf* node: t is a leaf of T;
– a node *introducing* a vertex $v \in V(G)$: t has a unique child t' in T, and $B_t = B_{t'} \cup \{v\}$;
– a node *forgetting* a vertex $v \in V(G)$: t has a unique child t' in T, and $B_t = B_{t'} \setminus \{v\}$;
– a *join* node: t has exactly two children, t' and t'', in T, and $B_t = B_{t'} = B_{t''}$.

2.2 Parameterized complexity

We want to investigate hard variants of the elimination problem in greater detail using the ideas of parameterized complexity. In this framework, we associate an integer ℓ called the *parameter* with each input I, and express the running time of an algorithm as a function of both the input size $|I|$ and the parameter ℓ, allowing us to measure the complexity of the problem in a two-dimensional fashion.

The aim in the study of a parameterized problem is to show that it is *fixed-parameter tractable* (or FPT for short), by giving an algorithm that runs in time $f(\ell) \cdot |I|^{O(1)}$ for some computable function f. Notice that the dependence on $|I|$ in the running time is only a polynomial of constant degree; such an algorithm is called a *fixed-parameter tractable algorithm* (FPT algorithm for short).

The basic class of parameterized intractability is W[1]; if a parameterized problem is *W[1]-hard*, then this yields strong evidence that it does not admit an FPT algorithm. One can prove W[1]-hardness by means of a *parameterized reduction* (also called *FPT reduction*). For more details on parameterized complexity, we refer the reader to the recent monograph by Downey and Fellows [8].

3 Polynomial-time solvable special cases

To obtain polynomial-time algorithms for $\textsc{alc}(S)$ applicable on certain classes of input instances, we shall first deal with trees.

Theorem 2 ALC(S) *can be solved in* $O(kn)$ *time on a graph* G *for which* $\bar{G}$ *is a tree on* n *vertices.*

Proof. Let $I = (G, c)$ be our input instance. We perform a standard-style dynamic programming on trees, so we assume that $T = \bar{G}$ is a rooted tree with root v_{root}. For any vertex $v \in V(G)$, we let $C_v \subseteq V(G)$ denote the set of children of v, T_v the subtree of T rooted at v, and $G_v = G[T_v]$.

We compute a non-negative value $s(v)$ for each $v \in V(G)$ defined as follows: we let $s(v)$ be the minimum score of v in any score assignment that is valid for G_v with respect to c. If no such assignment exists, we set $s(v) = +\infty$.

Clearly, $s(v) = 0$ for each leaf v of T. To compute $s(v)$ for a non-leaf node, we first compute for each $x \in C_v$ the minimum gain of v from the arc (v, x) (or (x, v)) in any score assignment for $G[T_x \cup \{v\}]$ that is valid on the vertices of T_x; we denote this value by $g_v(x)$. It is easy to see that

$$g_v(x) = \begin{cases} \min\{\alpha_q \mid q \in \{0, \ldots, k\}, s(x) + \beta_q \leq c(x)\} \text{ if } (v, x) \in A(G), \\ \min\{\beta_q \mid q \in \{0, \ldots, k\}, s(x) + \alpha_q \leq c(x)\} \text{ if } (x, v) \in A(G). \end{cases}$$

Notice that if the minimum is taken over an empty set in the above expression, then by $\alpha_0 = \beta_k = 0$ this means that no valid score assignment exists for G_x; in such a case we let $g_v(x) = +\infty$.

Knowing $g_v(x)$ for each child x of v, one can compute $s(v)$ as follows.

$$s(v) = \begin{cases} \sum_{x \in C_v} g_v(x) & \text{if } \sum_{x \in C_v} g_v(x) \leq c(v), \\ +\infty & \text{otherwise.} \end{cases}$$

We compute $s(v)$ in a bottom-up manner (e.g., by performing a DFS on T), so when computing $s(v)$, we assume that $s(x)$ is already known for each $x \in C_v$. By definition, (G, c) is solvable exactly if we obtain $s(v_{root}) \leq c(v_{root})$. The correctness of the algorithm easily follows from the above arguments.

Computing $g_v(x)$ for some $x \in C_v$ takes $O(k)$ time, so calculating $s(v)$ takes $O(k|C_v|)$ time. Altogether, the algorithm runs in $O(k|A(G)|) = O(kn)$ time. $\square$

Remark 1. Observe that the algorithm of Theorem 2 can find a valid score assignment minimizing the score of an arbitrary vertex of the input graph, by considering it as the root.

Remark 2. Notice that an instance $I = (G, c)$ of ALC(S) is solvable if and only if it is solvable for each connected component of G. Thus, Theorem 2 implies that ALC(S) can be solved in $O(kn)$ time for any graph G for which $\bar{G}$ is a forest on n vertices.

Let us also mention that Theorem 2 can be generalized to work for input graphs G for which $(\bar{G})^1$ is a forest. Such an algorithm is presented in Section 5, see Corollary 2.

Theorem 3 ALC(S) *is solvable in* $O(k^2 n)$ *time on a graph* G *for which* $\bar{G}$ *is a cycle on* n *vertices.*

Proof. The algorithm chooses any arc (v_1, v_2) of the cycle $\bar{G}$. For each outcome $(\alpha_i, \beta_i) \in S$, it tries to assign (α_i, β_i) to the arc (v_1, v_2), and checks if this leads to a valid score assignment. To do so, it decreases the capacity of v_1 and v_2 by α_i and β_i, respectively, and then calls the algorithm of Theorem 2 for the graph $G - (v_1, v_2)$ with the obtained capacities. As $\bar{G} - (v_1, v_2)$ is a path, the algorithm of Theorem 2 can indeed be used. Since there are $k + 1$ possibilities of assigning an outcome to (v_1, v_2), Theorem 2 yields that the overall running time is $O(k^2 n)$. □

To close this section, we observe that Theorems 2 and 3 lead to an efficient algorithm for 2-ALC(S). Notice that a polynomial algorithm for 2-ALC(S) with $S = \{(0, 3), (1, 1), (3, 0)\}$ (European Football scoring rules) has already been given by Bernholt et al. [4].

Theorem 4 2-ALC(S) *is solvable in* $O(k^2 n)$ *time on a graph with* n *vertices.*

Proof. If $I = (G, c)$ is an instance of 2-ALC(S), then $\bar{G}$ is a collection of vertex disjoint paths and cycles. Using Theorems 2 and 3 for each connected component of G separately, we obtain an overall computational complexity of $O(k^2 n)$. □

4 Complexity dependence on degree bounds

Considering instances of ALC(S) where the input graph $G = (V, A)$ has bounded degree, Theorem 4 and Corollary 1 indicate a strict threshold separating tractability from intractability: 2-ALC(S) is polynomial-time solvable, while 3-ALC(S) remains NP-complete. This leads to the following natural question: can 3-ALC(S) be solved efficiently if there are only a *few* vertices in G having degree three? We show the following assertion.

Theorem 5 3-ALC(S) *can be solved in time* $O((k + 1)^{d_3+2} n)$*, where* n *is the number of vertices in the input graph* G*, and* d_3 *is the number of vertices in* G *with degree three.*

Proof. The basic idea of the algorithm is the following: for each vertex of degree three, choose one incident arc, and assign to it an element from S. It is easy to see that there are at most $(k + 1)^{d_3}$ such assignments. By deleting all the chosen arcs and adjusting the capacities of the vertices appropriately, we obtain an equivalent instance of 2-ALC(S) that is solvable in $O(k^2 n)$ time. So the total running time of the algorithm is $O((k + 1)^{d_3+2} n)$. □

A straightforward generalization of the above approach is to study the complexity of ALC(S) in the case when there are only a few vertices in the input graph G that have degree more than two but, as opposed to the setting in Theorem 5, these vertices may have arbitrary degree. In terms of parameterized complexity, this motivates the investigation of the parameter $d_{>2}$ denoting the number of vertices in G with degree more than two. In the next theorem, we show an FPT algorithm for ALC(S) with parameter $d_{>2}$. Observe that while Theorem 6 obtains fixed-parameter tractability for a more general problem than the one considered in Theorem 5, the running time of the presented algorithm is considerably worse than the one obtained in Theorem 5.

In the proof of Theorem 6, we will find it convenient to consider the following generalization of ALC(S) where the set of allowed outcomes for each arc is restricted by certain constraints; we call this variant of the problem CONSTRAINED ARC LABELLING WITH CAPACITIES FOR S or CALC(S) for short. Let $\varphi : A \to 2^{\{0,\dots,k\}}$ be a function representing our constraints; we say that $\varphi(a)$ is the set of *allowed outcomes* for the arc $a \in A$. Score assignment p is said to *respect* φ if $p(a) \in \varphi(a)$ holds for each $a \in A$.

> **Constrained Arc Labelling with Capacities for** $S = \{(\alpha_0, \beta_0), \dots, (\alpha_k, \beta_k)\}$:
> *Instance:* A triple (G, c, φ) where $G = (V, A)$ is a directed multigraph, $c : V \to \mathbb{R}$ is a vertex capacity function, and $\varphi : A \to 2^{\{0,\dots,k\}}$ describes arc constraints.
> *Question:* Does there exist an assignment $p : A \to \{0, \dots, k\}$ respecting φ for which $\mathrm{scr}_p(v) \leq c(v)$ for each vertex $v \in V$?

Theorem 6 ALC(S) *is FPT with parameter $d_{>2}$ denoting the number of vertices in the input graph G with degree more than two.*

Proof. Let (G, c) be the input given for ALC(S), and let Z be the set of vertices in G with degree more than two.

Components with maximum degree at most 2. To begin, we apply the algorithm of Theorem 4 for 2-ALC(S) on each connected component of G that contains only vertices of degree at most two. After this preprocessing step, we may assume that G has no such connected component.

Structural observations. Let H be the graph $\bar{G} - Z$. Observe that H has maximum degree at most 2, and thus must be a disjoint union of cycles and paths. From our assumption on the connected components of $\bar{G}$, we get that each connected component of H must be adjacent to some vertex of Z in G. However, as each vertex of H has degree at most two in G, we know that any vertex of H adjacent to Z in G can have degree at most one in H. This means that H is a disjoint union of a set $\mathcal{P}$ of paths, with each path $P \in \mathcal{P}$ having at least one endpoint adjacent to Z. Clearly, the inner points of P cannot be adjacent to Z. Let $\mathcal{P}_1$ ($\mathcal{P}_2$) denote those paths in $\mathcal{P}$ that have one endpoint (two endpoints, respectively) adjacent to Z.

Paths with only one endpoint adjacent to Z. First, suppose that P is a path in $\mathcal{P}_1$, with one of its endpoints adjacent to some vertex z of Z. Let P_z denote the subgraph induced by $P \cup \{z\}$ in G; clearly, $\bar{P}_z$ is a path. We proceed by using the algorithm of Theorem 2 to find a valid score assignment for P_z minimizing the score of z (see Remark 1). In case this algorithm returns a valid score assignment on P_z with z having score $s_z \leq c(z)$, then we can delete P from G and decrease the capacity of z by s_z; this operation does not change the solvability of (G, c). In case no such assignment is found, then we know that there is no valid assignment for our instance (G, c), and we can stop. After dealing this way with each path in $\mathcal{P}_1$, we can assume that $\mathcal{P}_1 = \emptyset$.

Reduction to **CALC.** Our algorithm proceeds by constructing an equivalent instance (G', c', φ) of CALC(S') for a broader set

$$S' = \{(\gamma, \delta) \mid \gamma, \delta \in \cup_{i=0,\dots,k}\{\alpha_i, \beta_i\}\}$$

of outcomes. Note that $|S'| \leq 4(k+1)^2$. We construct G' and the constraints φ as follows. We let $V(G') = Z$ and $A(G') = A_Z \cup A_P$. Here, A_Z is the set of all arcs in $G[Z]$, and A_P is a newly introduced arc set containing an arc a_P for each $P \in \mathcal{P}_2$. We set $\varphi(a) = S$ for each arc $a \in A_Z$; that is, we allow the set S of outcomes on the arcs of $G[Z]$.

To define a_P and the corresponding set of allowed outcomes $\varphi(a_P)$, we need some further definitions. Let us fix $P \in \mathcal{P}_2$. We denote by e_P and f_P the two arcs connecting P and Z in G. Let x_P and y_P denote the vertices of Z incident to e_P and f_P, respectively; note that $x_P = y_P$ is possible. We let P' be the subgraph of G obtained from $G[V(P)]$ by appending the arcs e_P and f_P; notice that $\bar{P}'$ is a path or a cycle. Supposing that s is a score assignment on P' that assigns score s_x to x_P resulting from e_P and score s_y to y_P resulting from f_P, we let the *shadow* of s on P' be the pair (s_x, s_y). We let $S(P)$ be the set of shadows of all valid score assignments on P'. Note that $S(P) \subseteq S'$, and $S(P)$ can easily be computed in $O(k^2|V(P)|)$ time with the algorithms of Theorems 2 and 3. We define a_P as an arc leading from x_P to y_P, and we let $\varphi(a_P) = S(P)$. This finishes the definition of G' and φ; we set $c'(z) = c(z)$ for each $z \in Z$.

We claim that (G, c) is a yes-instance of ALC(S) if and only if (G', c', φ) is a yes-instance of CALC(S').

First, suppose that s is a valid score assignment on G with respect to c. We construct a score assignment s' for G' that is valid with respect to c' and respects φ. Namely, for each arc a in $G[Z]$, we let $s'(a) = s(a)$, and for each $P \in \mathcal{P}_2$, we let $s'(a_P)$ be the shadow of s on path P'. By the validity of s and the definition of a shadow, we have $s'(a_P) \in \varphi(a_P)$, hence s' respects φ. Note also that s' assigns the same score as s to each vertex $z \in Z$, showing the validity of s' for (G', c', φ).

For the other direction, suppose s' is a score assignment for G' that is valid with respect to c' and respects φ; we define a valid score assignment s for (G, c) as follows. Again, we let $s(a) = s'(a)$ for each arc a in $G[Z]$. For each path $P \in \mathcal{P}_2$, we let the restriction of s on P' be a valid score assignment on P' whose shadow is $s'(a_P)$; such an assignment exists by definition. Note that each arc of G is either in $G[Z]$ or in P' for some $P \in \mathcal{P}_2$, so now s is well-defined. It should also be clear that s is valid as well, proving our claim.

Solving CALC **with an ILP.** To finish our algorithm, it remains to solve the instance (G', c', φ) of CALC(S'). To this end, we define an ILP I_{calc} as follows. For each $u, v \in V(G')$, $R \subseteq S'$, and $(\alpha, \beta) \in R$, we introduce a non-negative variable $x_{u,v,R,(\alpha,\beta)}$ describing how many of the arcs leading from u to v with the set of allowed outcomes being R should be assigned the outcome (α, β). Furthermore, we compute the number of arcs in G' leading from u to v for which the set of allowed outcomes given by φ is R; we denote this number by $\mu_{u,v,R}$. The ILP I_{calc} contains the following constraints.

$$\sum_{(\alpha,\beta)\in R} x_{u,v,R,(\alpha,\beta)} = \mu(u,v,R) \quad \forall u,v \in V(G'), \forall R \subseteq S' \tag{3}$$

$$\sum_{u\in V(G')} \sum_{R\subseteq S'} \sum_{(\alpha,\beta)\in R} x_{v,u,R,(\alpha,\beta)}\alpha + x_{u,v,R,(\alpha,\beta)}\beta \le c'(v) \quad \forall v \in V(G') \tag{4}$$

Equations (3) ensure that each arc is assigned an outcome, and Inequalities (4) guarantee the validity of the obtained score assignment. By $|V(G')| = |Z| = d_{>2}$, $|S'| \le 4(k+1)^2$, and $\sum_{R\subseteq S'}|R| = |S'|2^{|S'|-1} \le (k+1)^2 2^{4(k+1)^2+1}$ we obtain that I_{calc} has at most $(d_{>2})^2(k+1)^2 2^{4(k+1)^2+1}$ variables. The number of constraints in I_{calc} is $(d_{>2})^2 2^{4(k+1)^2} + d_{>2}$. Altogether, we obtain that the size of I_{calc} is a function depending only on $d_{>2}$ and k. Hence, for each constant value of k, we have that $|I_{calc}|$ is upper bounded by $f(d_{>2})$ for some computable function f. Using, e.g., the algorithm of Lenstra [19], we can compute in $f'(d_{>2})$ time whether I_{calc} is solvable, for some computable function f'.

Running time analysis. Dealing with the connected components of G with maximum degree at most two can be done in $O(k^2|V(G)|)$. The step which takes care of paths in $\mathcal{P}_1$ needs time $O(k|V(G)|)$. The CALC(S') instance (G', c', φ) can be constructed in $O(k^2|V(G)|) + O(|(G,c)|)$ time where $|(G,c)|$ denotes the size of the ALC(S) instance (G,c). Finally, solving the ILP I_{calc} takes $f'(d_{>2})$ time. Altogether, the running time of our algorithm is $f'(d_{>2}) + O(|(G,c)|)$ for some computable function f', yielding a linear-time FPT algorithm with parameter $d_{>2}$. □

5 Complexity depending on feedback parameters

Since ALC(S) is polynomial-time solvable on trees, as shown in Theorem 2, it is interesting to examine how the complexity of the problem changes if we consider instances which are "almost trees". There are several graph classes which can be viewed as a generalization of trees, yielding numerous ways to approach this issue. One possibility is to consider graphs which admit a small feedback vertex or edge set, that is, graphs which can be turned into trees by deleting a few vertices or edges.

Using the terminology of parameterized complexity, we are interested in parameterizations where the parameter associated with an instance of ALC(S) is the feedback vertex set number or the feedback edge set number of the (undirected version of the) input graph G, that is, $\text{fvs}(\bar{G})$ or $\text{fes}(\bar{G})$. It is worth mentioning that the feedback edge set number of a graph can be determined in linear time (by computing a spanning tree for each connected component), while the feedback vertex set number can be computed in FPT time (see the work by Cao et al. [7] for the currently fastest deterministic algorithm) and 2-approximated in polynomial time [2, 3]. Hence, obtaining fixed-parameter tractability for either of these parameters would yield an algorithm that could be used efficiently in practice, if the parameter is small.

As it turns out, the complexity of these problems differs for the vertex and the edge variants: as Theorem 8 shows, ALC(S) is W[1]-hard when the parameter is $\text{fvs}(\bar{G})$, so under standard complexity-theoretic assumptions, we cannot expect an FPT algorithm for it. By contrast, one can easily construct an FPT algorithm for the parameter $\text{fes}(\bar{G})$.

Theorem 7 ALC(S) *is FPT if the parameter is* $\text{fes}(\bar{G})$ *for input graph* G.

Proof. Let F be a set of arcs in G corresponding to a minimum feedback edge set in $\bar{G}$. Then the problem can be solved by an algorithm whose main steps are as follows. First, we enumerate all possible score assignments on the arcs of F, and then, for each possibility, we check whether there is a score assignment on the remaining arcs of G which, together with the assignment on F, yields a valid score assignment for G.

There are at most $(k+1)^{|F|}$ possible score assignments on F, and the second step for each case can be performed in $O(k|V(G)|)$ time, using that $G-F$ (or, more precisely, its undirected version) is a forest. Thus, the algorithm runs in time $(k+1)^{|F|+1}O(|V(G)|)$, which is fixed-parameter tractable with parameter $|F|$. □

Theorem 8 *Except for the case where* S *is of the form* $\{(i,k-i) \mid 0 \le i \le k\}$ *for some* $k \in \mathbb{N}$, ALC(S) *is W[1]-hard if the parameter is* $\text{fvs}(\bar{G})$*, where* G *is the input graph.*

Proof. We present a parameterized reduction from the UNARY BIN PACKING problem. In this problem, we are given n items having sizes $s_1, \dots, s_n$, where each s_i is a positive integer encoded in unary, an integer bin capacity b, and an integer parameter m. The task is to decide whether the given items can be packed into m bins such that the total size of the items contained in any bin does not exceed the bin capacity b. This problem is known to be W[1]-hard if the parameter is the number m of bins [13].

Given an instance $I = (s_1, \dots, s_n, b, m)$ of UNARY BIN PACKING, we construct an equivalent instance (G, c) of ALC(S) in polynomial time such that the graph $\bar{G}$ has a feedback vertex set of size at most m. We distinguish three cases, depending on the set S of outcomes, but first we give some definitions useful for each case.

We define an undirected graph H as follows. For each $i = 1, \dots, n$, we fix an arbitrary rooted binary tree having exactly s_i leaves. We take m disjoint copies of each of these n trees, and denote by T_i^j the j-th copy of the i-th tree. We denote the root of T_i^j by r_i^j, and we write L_i^j for the set of leaves in T_i^j.

Next, for each $i = 1, \dots, n$ we introduce a star on $m+1$ vertices, with central vertex a_i, and leaves ${a'}_i^1, \dots, {a'}_i^m$. For each $j = 1, \dots, m$ we connect ${a'}_i^j$ to the root r_i^j. Similarly, for each $j = 1, \dots, m$ we introduce a star on $1 + \sum_{i=1}^n s_i$ vertices, with central vertex b_j and leaves partitioned into sets ${L'}_1^j, \dots, {L'}_n^j$, with

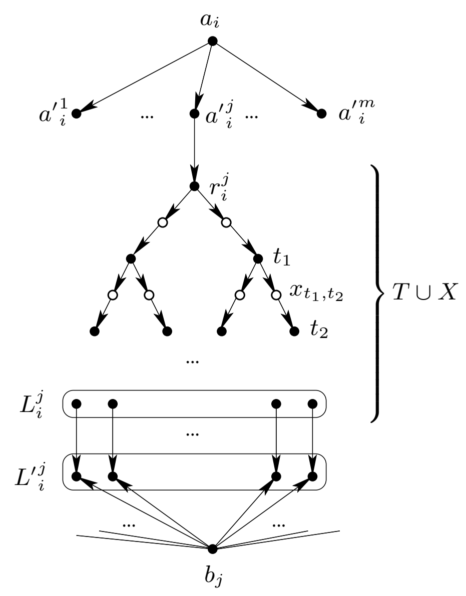

Fig. 1. Illustration of the graph G defined in Case A of Theorem 8. Vertices of X are shown as white circles.

the condition $|L'^j_i| = s_i$. For each i and j, we connect vertices in L^j_i and in L'^j_i by creating a perfect matching between them. This finishes the definition of H.

Intuitively, the vertex a_i corresponds to the item with size s_i, and the score assignment on T^j_i encodes whether we put this item into the j-th bin or not. The vertex b_j represents the j-th bin.

We use the notation $A = \{a_1, \dots, a_n\}$, $B = \{b_1, \dots, b_m\}$, $L = \cup_{i=1}^n \cup_{j=1}^m L^i_j$, $T = \cup_{i=1}^n \cup_{j=1}^m V(T^i_j)$, $A' = N(A)$, and $L' = N(B)$. Also, for some $\ell \in L^j_i$ we write ℓ' to denote its unique neighbor in L'.

Case A: "large gap on the β-side". First, let us deal with the case where $\beta_{q-1} > \beta_q + 1$ for some $1 \le q \le k$.

We define a directed graph G obtained from H as follows; see Figure 1. First, we orient the edges not contained in $H[T]$ such that the vertices of A and B become sources, the vertices of L' become sinks, and each vertex of A' has in-degree and out-degree exactly 1 (that is, a'^j_i receives an arc from a_i and sends an arc to r^j_i). Second, for each edge $\{t_1, t_2\}$ within some tree T^j_i, with t_1 being the parent of t_2, we replace $\{t_1, t_2\}$ by a directed path (t_1, x_{t_1,t_2}, t_2) of length 2. We denote by X the set of vertices of the type x_{t_1,t_2} introduced by such replacements.

Let us observe that deleting the vertices of B from $\bar{G}$ yields a forest consisting of n trees, so the size of a minimum feedback vertex set for $\bar{G}$ is at most m, as promised. Moreover, the size of G is polynomial in the size of the instance I, because each integer s_i is encoded in unary.

To finish the reduction, it remains to define the capacity function c. We let

$$\begin{array}{ll}
c(a_i) = m-1 & \text{for each } a_i \in A, \\
c(b_j) = b & \text{for each } b_j \in B, \\
c(t) = \max\{\beta_{q-1}, \beta_q + 2\} & \text{for each } t \in T \setminus L, \\
c(\ell) = \beta_{q-1} & \text{for each } \ell \in L, \\
c(v) = \max\{\alpha_{q-1} + \beta_0, \alpha_q + \beta_1\} & \text{for each } v \in A' \cup X, \text{ and} \\
c(\ell') = \beta_0 + \beta_1 & \text{for each } \ell' \in L'.
\end{array}$$

To prove the correctness of the reduction, we show that I is a yes-instance of UNARY BIN PACKING if and only if (G, c) is a yes-instance of ALC(S).

"$\Longrightarrow$": Let us first suppose that I is a yes-instance of UNARY BIN PACKING, i.e., there exists a mapping $\varphi : \{1, \dots, n\} \to \{1, \dots, m\}$ representing the packing of the items into the given bins such that $\sum_{i:\varphi(i)=j} s_i \leq b$ holds for each $j = 1, \dots, m$. Then we can define a score assignment $p : A(G) \to \{0, \dots, k\}$ for G that is valid with respect to c as follows. (Indices i and j take all possible values meeting the specified conditions.)

$$p(a_i {a'}_i^j) = \begin{cases} 0, & \text{if } j = \varphi(i), \\ 1, & \text{otherwise.} \end{cases}$$

$$p(vt) = \begin{cases} q-1, & \text{if } (v,t) \in A(G) \text{ and } t \in T_i^j \text{ with } j = \varphi(i), \\ q, & \text{if } (v,t) \in A(G) \text{ and } t \in T_i^j \text{ with } j \neq \varphi(i). \end{cases}$$

$$p(tv) = \begin{cases} 0, & \text{if } (t,v) \in A(G) \text{ and } t \in T_i^j \text{ with } j = \varphi(i), \\ 1, & \text{if } (t,v) \in A(G) \text{ and } t \in T_i^j \text{ with } j \neq \varphi(i). \end{cases}$$

$$p(b_j \ell') = \begin{cases} 1, & \text{if } \ell' \in {L'}_i^j \text{ with } j = \varphi(i), \\ 0, & \text{if } \ell' \in {L'}_i^j \text{ with } j \neq \varphi(i). \end{cases}$$

It is straightforward to check that p is indeed a valid score assignment for G and c.

"$\Longleftarrow$": For the reverse direction, let us assume that p is a valid score assignment for G with respect to the given capacities. Let us fix some $i \in \{1, \dots, n\}$. We define an index $\varphi(i)$, depending on p, such that putting the item of size s_i into the bin having number $\varphi(i)$ for each i yields a solution for I.

By $c(a_i) = m-1$ we know that p must assign the outcome $(0, \beta_0)$ to at least one arc leaving a_i, as $\alpha_j \geq 1$ for any $j \geq 1$. So let $(a_i, {a'}_i^j)$ be an arc which is assigned the outcome $(0, \beta_0)$, that is, $p(a_i {a'}_i^j) = 0$. We define $\varphi(i) = j$.

Observe that $c({a'}_i^j) < \alpha_q + \beta_0$. As ${a'}_i^j$ already gets β_0 points resulting from the arc $(a_i, {a'}_i^j)$, we get that ${a'}_i^j$ must gain less than α_q scores resulting from the arc $({a'}_i^j, r_i^j)$, implying $p({a'}_i^j r_i^j) \leq q-1$. Hence, r_i^j gains at least β_{q-1} points because of this arc. Note that $c(r_i^j) < \beta_{q-1} + 1$ follows from our assumption $\beta_{q-1} > \beta_q + 1$, which leads to the consequence that r_i^j gains exactly β_{q-1}, and both arcs leaving r_i^j must be assigned the outcome $(0, \beta_0)$ in p.

Applying these arguments repeatedly, we arrive to the following: p must assign the outcome $(0, \beta_0)$ to each arc leaving a vertex of $T_i^j \setminus L$, and each $t \in T_i^j$

obtains β_{q-1} points resulting from the unique arc entering t. In particular, each leaf $\ell \in L_i^j$ must obtain exactly β_{q-1} points, meaning also that the arc (ℓ, ℓ') must be assigned the outcome $(0, \beta_0)$. Hence, each $\ell' \in {L'}_i^j$ may gain at most β_1 points resulting from the arc (b_j, ℓ'), yielding that b_j gets at least 1 point from each arc leading to some vertex of ${L'}_i^j$. By $|{L'}_i^j| = s_i$, this means that b_j gains a total of at least s_i points from such arcs.

Taking into account all indices i for which $\varphi(i) = j$, we get that b_j gains at least $\sum_{i:\varphi(i)=j} s_i$ points in total. As each b_j has capacity b, we get that $\sum_{i:\varphi(i)=j} s_i \leq b$ holds for each $j = 1, \ldots, m$, which means exactly that packing the item of size s_i into the j-th bin for $j = \varphi(i)$ is a solution for our instance I of UNARY BIN PACKING. This finishes the proof of correctness for our reduction in Case A.

Case B: "small gap on the β-side". Let us now deal with the case where there is an index q, $1 \leq q \leq k$ for which $\beta_{q-1} < \beta_q + 1$; if there are several such indices, we let q be the smallest one. As we can also assume that the condition of Case A does not hold, we have $\beta_{j-1} = \beta_j + 1$ for each $1 \leq j < q$.

To define the constructed instance (G, c) for this case, we first create G from H as follows; see Figure 2. First, we orient all edges not contained in $H[T]$ such that vertices of A and B become sources, while vertices of A' and L' become sinks. Second, for each edge $\{t_1, t_2\}$ within some tree T_i^j, with t_1 being the parent of t_2, we replace $\{t_1, t_2\}$ by newly introduced vertices x_{t_1,t_2}, y_{t_1,t_2}, and z_{t_1,t_2} together with the arcs (x_{t_1,t_2}, t_1), $(x_{t_1,t_2}, y_{t_1,t_2})$, $(y_{t_1,t_2}, z_{t_1,t_2})$, and (t_2, z_{t_1,t_2}). We write X, Y, and Z for the set of vertices of the type x_{t_1,t_2}, y_{t_1,t_2}, and z_{t_1,t_2}, respectively.

Notice that G can again be constructed in polynomial time, and $\bar{G}$ again has a feedback vertex set of size m, namely B. To finish the construction, we define vertex capacities as follows.

$$\begin{array}{ll}
c(a_i) = m - 1 & \text{for each } a_i \in A, \\
c(b_j) = b & \text{for each } b_j \in B, \\
c(t) = \max\{2\beta_{q-1}, 1 + 2\beta_q\} & \text{for each } t \in T \setminus L, \\
c(\ell) = 1 & \text{for each } \ell \in L, \\
c(x) = \alpha_{q-1} + \alpha_q & \text{for each } x \in X, \\
c(y) = \beta_q + 1 & \text{for each } y \in Y, \text{ and} \\
c(v) = \beta_0 + \beta_1 & \text{for each } v \in A' \cup L' \cup Z.
\end{array}$$

Next we prove that I is a yes-instance of UNARY BIN PACKING if and only if (G, c) is a yes-instance of ALC(S).

"$\Longrightarrow$": First suppose that I is a yes-instance of UNARY BIN PACKING, i.e., there exists a mapping $\varphi : \{1, \ldots, n\} \to \{1, \ldots, m\}$ representing the packing of the items into the given bins such that $\sum_{i:\varphi(i)=j} s_i \leq b$ holds for each $j = 1, \ldots, m$. We define a score assignment $p : A(G) \to \{0, \ldots, k\}$ for G that is valid with respect to c as follows.

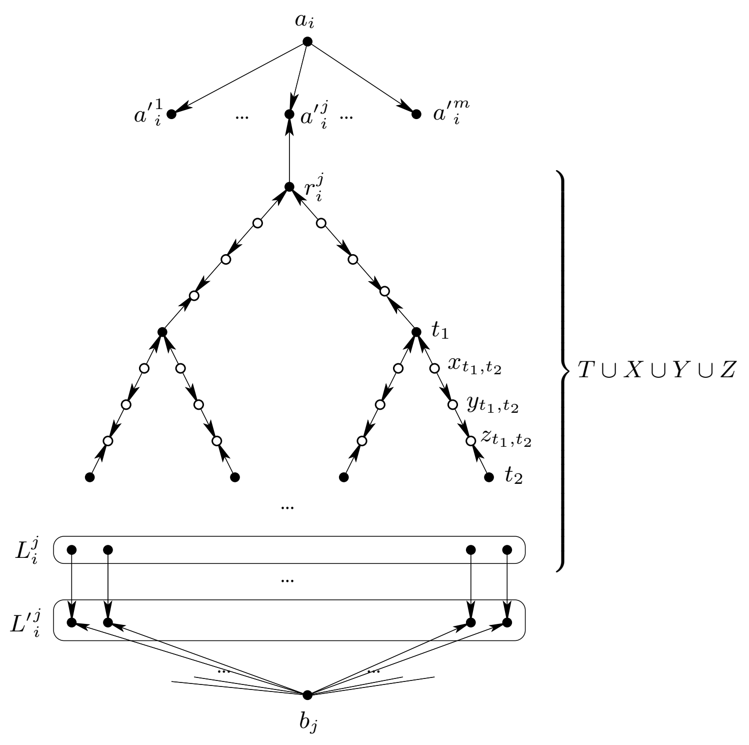

Fig. 2. Illustration of the graph G defined in Case B of Theorem 8. Vertices of $X \cup Y \cup Z$ are shown as white circles.

$$p(a_i a'^j_i) = \begin{cases} 0, & \text{if } j = \varphi(i), \\ 1, & \text{otherwise.} \end{cases}$$

$$p(tv) = \begin{cases} 1, & \text{if } v \in A' \cup Z,\ (t,v) \in A(G), \text{ and } t \in T^j_i \text{ with } j = \varphi(i), \\ 0, & \text{if } v \in A' \cup Z,\ (t,v) \in A(G), \text{ and } t \in T^j_i \text{ with } j \neq \varphi(i). \end{cases}$$

$$p(xt) = \begin{cases} q, & \text{if } x \in X,\ (x,t) \in A(G), \text{ and } t \in T^j_i \text{ with } j = \varphi(i), \\ q-1, & \text{if } x \in X,\ (x,t) \in A(G) \text{ and } t \in T^j_i \text{ with } j \neq \varphi(i). \end{cases}$$

$$p(x_{t,t'} y_{t,t'}) = \begin{cases} q-1, & \text{if } \{t,t'\} \in E(T^j_i) \text{ with } j = \varphi(i), \\ q, & \text{if } \{t,t'\} \in E(T^j_i) \text{ with } j \neq \varphi(i). \end{cases}$$

$$p(y_{t,t'} z_{t,t'}) = \begin{cases} 0, & \text{if } \{t,t'\} \in E(T^j_i) \text{ with } j = \varphi(i), \\ 1, & \text{if } \{t,t'\} \in E(T^j_i) \text{ with } j \neq \varphi(i). \end{cases}$$

$$p(\ell\ell') = \begin{cases} 0, & \text{if } \ell \in L^j_i \text{ with } j = \varphi(i), \\ 1, & \text{if } \ell \in L^j_i \text{ with } j \neq \varphi(i). \end{cases}$$

$$p(b_j \ell') = \begin{cases} 1, & \text{if } \ell' \in L'^j_i \text{ with } j = \varphi(i), \\ 0, & \text{if } \ell' \in L'^j_i \text{ with } j \neq \varphi(i). \end{cases}$$

Again, it is straightforward to verify the validity of p for G and c.

"$\Longleftarrow$": Suppose now that p is a valid score assignment for G and c. First, let us observe that by the choice of q, we have $\beta_{q-2} = \beta_{q-1} + 1 > \beta_q + 1 = c(y)$ for each $y \in Y$. Therefore, we get that $p(xy) \geq q - 1$ for each $x \in X$ and $y \in Y$ with $(x, y) \in A(G)$. Thus, x gains at least α_{q-1} scores resulting from the arc (x, y), so by $c(x) = \alpha_q + \alpha_{q-1}$ we know that $p(xt) \leq q$ must hold for the other arc (x, t) leaving x. This shows that each vertex $t \in T \setminus L$ gains at least β_q points on both of its incoming arcs.

Now, let us fix some i with $1 \leq i \leq n$. Arguing the same way as for Case A, we can define an index $\varphi(i) = j$ such that $p(a_i {a'}^j_i) = 0$. By $c({a'}^j_i) = \beta_0 + \beta_1$, we get that $p(r^j_i {a'}^j_i) \geq 1$ and therefore, r^j_i gains at least 1 point resulting from its outgoing arc. By the previous paragraph, it also gains at least $2\beta_q$ points resulting from its incoming arcs. Note also that $c(r^j_i) < 1 + \beta_q + \beta_{q-1}$ follows from our assumption $\beta_{q-1} < \beta_q + 1$, yielding that p can only assign the outcome (α_q, β_q) for both arcs entering r^j_i.

By slight abuse of the notation, let us write $r = r^j_i$ now, and let $x_{r,t}$ be an in-neighbor of r^j_i (so t is a child of $r = r^j_i$ in T^j_i). Using that $p(x_{r,t} r) = q$, the capacity bounds now imply $p(x_{r,t} y_{r,t}) = q - 1$ and $p(y_{r,t} z_{r,t}) = 0$. By $c(z_{r,t}) = \beta_0 + \beta_1$, it follows that $p(t z_{r,t}) \geq 1$. This means that t gains at least 1 point from its outgoing arc. Arguing repeatedly the same way as before, we arrive to the fact that each vertex of T^j_i must gain at least 1 point resulting from its outgoing arc, leading to a vertex of $Z \cup A'$. In particular, all leaves $\ell \in L^j_i$ must gain exactly 1 point this way, implying $p(\ell \ell') = 0$, which in turn proves $p(b_j \ell') \geq 1$. Therefore, b_j gains at least 1 point resulting from each of its outgoing arcs leading to some vertex of ${L'}^j_i$. By $|{L'}^j_i| = s_i$, this means that b_j gains a total of s_i points from such arcs.

Using this observation and arguing the same way as in Case A, one can easily check that packing the item of size s_i into the j-th bin for $j = \varphi(i)$ gives a solution for our instance I of UNARY BIN PACKING.

Case C: "non-uniform gap on the α-side." Assume that Cases A and B do not hold. In this case, we have $\beta_q = \beta_{q-1} + 1$ for each $1 \leq q \leq k$. Since S is not of the form $\{(i, k - i) \mid 0 \leq i \leq k\}$, we know that $\alpha_q - \alpha_{q-1}$ is either larger than 1 or smaller than 1 (but positive) for some q. Let us define the set of reversed outcomes $S_r = \{(\beta, \alpha) \mid (\alpha, \beta) \in S\}$. By the above discussion, S_r fulfills the conditions of Case A or B. Let f_{S_r} denote the reduction proving the W[1]-hardness of ALC(S_r), described in the case applicable for S_r. Thus, for any instance I of UNARY BIN PACKING, $f_{S_r}(I)$ is an equivalent instance of ALC(S_r).

Now, let g denote the function that reverses the graph underlying an instance of ALC(S_r), that is, $g((G, c)) = (G_r, c)$ where G_r is obtained by reversing every arc in G. Clearly, (G, c) is a yes-instance of ALC(S_r) if and only if (G_r, c) is a yes-instance of ALC(S). To finish our proof, it suffices to observe that $g \circ f$ is a reduction proving the W[1]-hardness of ALC(S). □

An interesting question left open is whether taking the parameter to be $\text{fes}((\bar{G})^1)$, that is, the minimum size of a feedback edge set of the undirected and *simple* version of the input graph, yields fixed-parameter tractability. Observe that $\mathrm{fes}(\bar{G})$ can be arbitrarily large even if $\mathrm{fes}((\bar{G})^1)$ is bounded by some constant, as an edge of $(\bar{G})^1$ may represent an arbitrary number of parallel edges in $\bar{G}$. Hence, the FPT result for the parameter $\mathrm{fes}(\bar{G})$, presented in Theorem 7, does not yield fixed-parameter tractability for the parameter $\mathrm{fes}((\bar{G})^1)$.

6 Complexity depending on pathwidth and treewidth

In this section we concentrate on classical width parameters that measure how tree-like a graph is; we focus on the notions treewidth and pathwidth. Our aim is to investigate whether we can solve $\textsc{alc}(S)$ on graphs that have small treewidth (or even small pathwidth). As $\textsc{alc}(S)$ is polynomial-time solvable on forests by Theorem 2, we can hope that such instances might be tractable.

In Theorem 9 below, we present an algorithm for $\textsc{alc}(S)$ that runs in polynomial time if the (undirected and simple version of the) input graph has treewidth at most some fixed integer w. In the language of parameterized complexity, the theorem asserts that the problem is in XP when parameterized by treewidth. Note that degree of the polynomial in the running time claimed by Theorem 9 depends on our bound w on the treewidth. As we will see later, this dependence is unavoidable.

Theorem 9 *There exists an algorithm that, given an instance $I = (G, c)$ of $\textsc{alc}(S)$ where $\bar{G}$ has treewidth at most w, solves I in $w^{O(w)}|I|^{O(w^2)}$ time.*

Proof. Observe that loops in the input graph can be treated easily, because we can assign to each loop the outcome (α_h, β_h) that minimizes $\alpha_h + \beta_h$ over all $h \in \{0, \dots, k\}$. Therefore, w.l.o.g. we assume that G has no loops.

Our algorithm uses standard-style dynamic programming based on tree-decomposition. First we compute a nice tree-decomposition of width at most $5w + 4$ for $\bar{G}$ using the 5-approximation algorithm for treewidth by Bodlaender et al. [6]; let $w' \leq 5w + 4$ be the width of the tree-decomposition obtained. Next, we transform this tree-decomposition into a nice one without increasing its width, using the algorithm described by Kloks [17]. Let $\mathbb{T} = (T, (B_t)_{t \in V(T)})$ be the tree-decomposition obtained. Clearly, we may assume that T is rooted at a node r for which $|B_r| = 1$; we can achieve this by possibly adding at most w' new nodes "above" r, each forgetting a vertex in B_r. For each $t \in V(T)$, we define $V_t = \bigcup \{B_{t'} \mid t' \text{ is a descendant of } t\}$, and we let G_t be the graph obtained from $G[V_t]$ by deleting all arcs connecting vertices of B_t.

Now, let us compute a set $U \subseteq \mathbb{R}$ which contains all possible score values that a vertex in V can take under any score assignment. In particular, we let U contain all numbers of the form $p_0^+\alpha_0 + \dots + p_k^+\alpha_k + p_0^-\beta_0 + \dots + p_k^-\beta_k$ where $p_0^+, \dots, p_k^+$ and $p_0^-, \dots, p_k^-$ are non-negative integers summing up to at most the maximum degree Δ in G. Notice that U has size at most $\Delta^{2(k+1)}$ and can be computed in $O(\Delta^{2(k+1)})$ time.

For each node $t \in V(T)$ with $B_t = \{v_1, \dots, v_\tau\}$, we compute a certain set S_t. To define S_t, first let S_t^* contain those score vectors $(s_1, \dots, s_\tau) \in U^\tau$ for which there exists a valid[5] score assignment for G_t with v_i having score s_i, $i = 1, \dots, \tau$. Now, we define S_t as the "minimal" score vectors in S_t, that is, we put $s = (s_1, \dots, s_\tau) \in S_t^*$ into S_t exactly if there is no vector $s' = (s'_1, \dots, s'_\tau) \in S_t^*$ with $s' < s$, that is, if $s'_i \leq s_i$ for each $i = 1, \dots, \tau$ but $s' \neq s$.

We compute S_t for each node $t \in V(T)$ in a bottom-up manner, using dynamic programming. Observe that since $|B_r| = 1$ and G has no loops, we have $G = G_r$, so there is a valid score assignment for G if and only if $S_r \neq \emptyset$. Let us describe our computations in detail, depending on the type of t.

Leaf node. Suppose that t is a leaf in T. Then G_t is a graph having no arcs, for which the "empty" score assignment is valid. Thus, S_t only contains the zero vector of length $|B_t|$.

Introduce node. Suppose now that t is a node introducing vertex v, and let t' be the unique child of t in T. Let $B_t = \{v, v_1, \dots, v_\tau\}$ and $B_{t'} = \{v_1, \dots, v_\tau\}$. Observe that G_t can be obtained from $G_{t'}$ by adding v as an isolated vertex. Thus, any valid score assignment for $G_{t'}$ is a valid score assignment for G_t as well, with v receiving score 0. Also, v receives score 0 in any score assignment for G_t. This implies $S_t = \{(0, s_1, \dots, s_\tau) \mid (s_1, \dots, s_\tau) \in S_{t'}\}$.

Join node. Suppose now that t is a join node with children t' and t''. Note that $V(G_{t'}) \cap V(G_{t''}) = B_t$, and observe that G_t can be obtained from $G_{t'}$ and $G_{t''}$ by identifying the two copies of each vertex in B_t. In particular, each arc of G_t is an arc either in $G_{t'}$ or in $G_{t''}$, but not in both. Hence, any valid score assignment p for G_t must be the union of a valid score assignment p' for $G_{t'}$ and a valid score assignment p'' for $G_{t''}$ in the sense that p coincides with p' on the arcs of $G_{t'}$ and coincides with p'' on the arcs of $G_{t''}$. For the converse direction, observe that valid score assignments p' and p'' for $G_{t'}$ and $G_{t''}$, respectively, combined this way yield a valid score assignment for G_t if and only if for any $v \in B_t$ the total score of v will not exceed its capacity.

These arguments lead us to the following way to compute S_t. First, we take all score vectors s which can be obtained as the sum of s' and s'' for some $s' \in S_{t'}$ and $s'' \in S_{t''}$. To obtain S_t we throw away all score vectors s from this set which are either non-minimal (i.e., for which there exists some $s' < s$ in the set) or which assign a score greater than $c(v)$ to some vertex $v \in B_t$.

By definition, each of S_t, $S_{t'}$, and $S_{t''}$ has cardinality at most $|U|^{w'+1}$. Therefore, the above computations take time at most $O(|U|^{2(w'+1)})$.

Forget node. Let t be a node forgetting some vertex v, and let t' be the unique child node of t. Let the vertices of $B_{t'}$ be $v, v_1, \dots, v_\tau$; we have $B_t = B_{t'} \setminus \{v\}$ and $\tau \leq w'$. Observe that G_t can be obtained from $G_{t'}$ by adding all arcs of the form (v, v_i) or (v_i, v) for each $v_i \in B_t$ that is present in G.

To compute S_t, we first compute the set S_t^*, from which S_t can easily be computed by getting rid of all non-minimal elements. For each possible score vector $s = (s_1, \dots, s_\tau) \in |U|^\tau$ we will solve several integer linear programs (ILPs) to decide whether there is a valid score assignment for G_t where v_i has score s_i, for each $i = 1, \dots, \tau$. Clearly, it suffices to consider s only if $s_i \leq c(v_i)$ holds for

[5] In this proof, we always consider validity with respect to the capacity function c, sometimes restricted to a subset of the vertices.

each i. For each such s, we check each score vector $s' = (s'_0, \ldots, s'_\tau) \in S_{t'}$, and examine if s can be obtained from s' in some specific sense. Namely, we check whether there exists a score assignment p for G_t yielding score vector s for the vertices of B_t such that the restriction of p on $G_{t'}$ yields score vector s' for the vertices of $B_{t'}$. Thus, let us now consider s and s' as fixed, and define an ILP $I_{s,s'}$ that is satisfiable if and only if these conditions hold for s and s'.

First, for each $v_i \in B_t$ and for each $h = 0, \ldots, k$, we introduce a non-negative variable $x_{\to i,h}$ to describe how many of the remaining (v, v_i)-matches result in the outcome (α_h, β_h). Similarly, we let a non-negative variable $x_{\leftarrow i,h}$ describe the number of remaining (v_i, v)-matches resulting in the outcome (α_h, β_h). Hence, our ILP $I_{s,s'}$ has $2(k+1)\tau$ non-negative integer variables, and it consists of the following constraints.

$$\sum_{h=0}^{k} x_{\to i,h} = \mu(v, v_i) \quad \text{for each } i = 1, \ldots, \tau, \tag{5}$$

$$\sum_{h=0}^{k} x_{\leftarrow i,h} = \mu(v_i, v) \quad \text{for each } i = 1, \ldots, \tau, \tag{6}$$

$$s'_i + \sum_{h=0}^{k} x_{\to i,h}\beta_h + \sum_{h=0}^{k} x_{\leftarrow i,h}\alpha_h \le s_i \qquad \text{for each } i = 1, \ldots, \tau, \tag{7}$$

$$s'_0 + \sum_{i=1}^{\tau} \left(\sum_{h=0}^{k} x_{\to i,h}\alpha_h + \sum_{h=0}^{k} x_{\leftarrow i,h}\beta_h \right) \le c(v). \tag{8}$$

Here, $\mu(x, y)$ for some $x, y \in V$ is the multiplicity of the arc (x, y) in G. It is easy to see that Equations (5) and (6) ensure that all remaining (v, v_i)-matches and (v, v_i)-matches receive an outcome. Inequalities (7) ensure that the scores of the vertices in B_t are as required by s, while Inequality (8) guarantees that the score of v does not exceed its capacity. It is straightforward to verify that $I_{s,s'}$ indeed checks the desired properties of s and s' as promised, implying that $s \in S^*_t$ holds exactly if $I_{s,s'}$ is satisfiable for some $s' \in S_{t'}$.

Let us analyze the running time necessary for this step. Notice that $I_{s,s'}$ has at most $2(k+1)w'$ variables and at most $3w' + 1$ constraints, and has total size $O(|I|)$. By using Kannan's improvement [14] on a result by Lenstra [19] for solving ILPs with a constant number of variables, we know that $I_{s,s'}$ can be solved in $(kw')^{O(kw')}|I_{s,s'}| = w'^{O(w')}|I|$ time. Note also that there are at most $|S_{t'}| \cdot |U|^\tau \le |U|^{(w'+1)w'}$ ILPs to solve, which yields an overall running time of $w'^{O(w')}|U|^{(w'+1)w'}|I|$ for computing S_t.

Running time. Given a graph G of treewidth at most w, the algorithm by Bodlaender et al. [6] finds a tree-decomposition for G with width at most $5w + 4$ in $2^{O(w)}|V(G)|$ time. This tree-decomposition can be turned into a nice one in linear time using the algorithm described by Kloks [17]. The time needed at some node of the tree-decomposition is dominated by the case of a forget node. Therefore, using $|U| \le \Delta^{2k+2} \le |I|^{2k+2}$ and that k is a fixed constant and $w' = O(w)$, we get that computing S_t for any node $t \in V(T)$ takes time

$w^{O(w)}|I|^{O(w^2)}$. Since T has size linear in the size of G, this leads to a total running time of $w^{O(w)}|I|^{O(w^2)}$. $\square$

As forests have treewidth 1, we have the following corollary.

Corollary 2 ALC(S) *can be solved in polynomial time if the input graph G is such that $\bar{G}^1$ is a forest.*

Theorem 9 shows that ALC(S) is polynomial-time solvable if the treewidth w of the underlying graph is bounded by a fixed constant. However, the degree of the polynomial describing the running time of the presented algorithm has a quadratic dependence on the treewidth w. Hence, it is a natural question to ask whether ALC(S) can be solved by an algorithm running in time $f(w)|I|^{O(1)}$ for some function f, or in other words, whether ALC(S) is fixed-parameter tractable with the parameter w.

The following theorem shows that this is unlikely, as the problem is W[1]-hard even if parameterized by the pathwidth of the underlying graph. This result implies W[1]-hardness for the parameter w as well, as stated in Corollary 3.

Theorem 10 *Except for the case where S is of the form $\{(i, k-i) \mid 0 \leq i \leq k\}$ for some $k \in \mathbb{N}$,* ALC(S) *is W[1]-hard when parameterized by the pathwidth of $\bar{G}$ where G is the input graph.*

Proof. The proof is very similar to the proof of Theorem 8. Again, we present a parameterized reduction from UNARY BIN PACKING. Let $I = (s_1, \dots, s_n, b, m)$ be the given input instance of UNARY BIN PACKING; recall that $s_1, \dots, s_n$ are integer item sizes (encoded in unary), b is the bin capacity, and m is the parameter, representing the number of bins. We construct an equivalent instance (G', c) of ALC(S) in polynomial time such that $\bar{G}'$ has pathwidth at most $m+3$, yielding a parameterized reduction.

We reuse some notation and definitions used in the proof of Theorem 8. In particular, we distinguish between the same three cases. Also, we make use of the graphs and vertex sets defined in the proof of Theorem 8.

Case A: "large gap on the β-side". Suppose $\beta_{q-1} > \beta_q + 1$ for some $1 \leq q \leq k$.

Using the graph G defined in Case A of Theorem 8, we define a graph G' obtained from G by replacing the subgraph of G induced by $T \cup X$ in a certain way. To this end, we first define a graph P_i^j for each $i = 1, \dots, n$ and $j = 1, \dots, m$ as follows. The vertex set of P_i^j consists of vertices $p_i^j(1), p_i^j(2), \dots, p_i^j(s_i - 1)$, and there are exactly s parallel arcs leading from $p_i^j(s-1)$ to $p_i^j(s)$, for each $s = 2, \dots, s_i - 1$. Next, we replace each arc of P_i^j similarly as in the proof of Theorem 8: the r-th arc leading from $p_i^j(s-1)$ to $p_i^j(s)$ is replaced by a vertex $x_i^j(r, s)$ together with arcs $(p_i^j(s-1), x_i^j(r, s))$ and $(x_i^j(r, s), p_i^j(s))$. We use X to denote the set of vertices introduced by these subdivisions. We denote the obtained directed graph by Q_i^j.

Now, we are ready to define G', obtained from $G - (T \cup X)$ as follows. For each i and j, we take Q_j^i, add an arc from a'^j_i to $p_i^j(1)$ and add one arc from

$p_i^j(s_i-1)$ to each $\ell' \in L'^j_i$, which means a total of s_i arcs leaving $p_i^j(s_i-1)$. It remains to define our capacity function c.

$$\begin{array}{ll}
c(a_i) = m-1 & \text{for each } a_i \in A, \\
c(b_j) = b & \text{for each } b_j \in B, \\
c(v) = \max\{\alpha_{q-1}+\beta_0, \alpha_q+\beta_1\} & \text{for each } v \in A' \cup X, \\
c(\ell') = \beta_0+\beta_1 & \text{for each } \ell' \in L', \text{ and} \\
c(p_i^j(s)) = \max\{s\beta_{q-1}, s\beta_q+s+1\} & \text{for each } p_i^j(s) \in V(P_i^j).
\end{array}$$

It is easy to see that the graph $\bar{G}'-B$ consists of n connected components, each having pathwidth 3, so $\bar{G}'-B$ itself has also pathwidth 3. By $|B|=m$, we obtain that $\bar{G}'$ indeed has pathwidth at most $m+3$, as promised. Now, we sketch the proof of correctness for our reduction showing that I is a yes-instance of UNARY BIN PACKING if and only if (G',c) is a yes-instance of ALC(S).

"$\Longrightarrow$": Suppose first that I is a yes-instance of UNARY BIN PACKING, i.e., there exists a mapping $\varphi : \{1,\dots,n\} \to \{1,\dots,m\}$ representing the packing of the items into the given bins such that $\sum_{i:\varphi(i)=j} s_i \le b$ holds for each $j = 1,\dots,m$. We define a score assignment $p : A(G') \to \{0,\dots,k\}$ for G' that is valid with respect to c as follows.

$$p(a_i a'^j_i) = \begin{cases} 0, & \text{if } j=\varphi(i), \\ 1, & \text{otherwise.} \end{cases}$$

$$p(vt) = \begin{cases} q-1, & \text{if } (v,t) \in A(G') \text{ and } t \in V(P_i^j) \text{ with } j=\varphi(i), \\ q, & \text{if } (v,t) \in A(G') \text{ and } t \in V(P_i^j) \text{ with } j\neq\varphi(i). \end{cases}$$

$$p(tv) = \begin{cases} 0, & \text{if } (t,v) \in A(G') \text{ and } t \in V(P_i^j) \text{ with } j=\varphi(i), \\ 1, & \text{if } (t,v) \in A(G') \text{ and } t \in V(P_i^j) \text{ with } j\neq\varphi(i). \end{cases}$$

$$p(b_j\ell') = \begin{cases} 1, & \text{if } \ell' \in L'^j_i \text{ with } j=\varphi(i), \\ 0, & \text{if } \ell' \in L'^j_i \text{ with } j\neq\varphi(i). \end{cases}$$

It is straightforward to check that p is a valid score assignment for G' and c.

"$\Longleftarrow$": For the reverse direction, let us assume that p is a valid score assignment for G with respect to the given capacities. Let us fix some $i \in \{1,\dots,n\}$. We define an index $\varphi(i)$, depending on p, such that putting the item of size s_i into the bin having number $\varphi(i)$ for each i yields a solution for I.

Following the argumentation of the proof of Theorem 8, we can again define $\varphi(i)$ as an index j for which $p(a_i a'^j_i)=0$. By the capacity of a'^j_i that we also have $p(a'^j_i p_i^j(1)) \le q-1$, so $p_i^j(1)$ gains at least β_{q-1} points resulting from its incoming arc. By our assumption $\beta_{q-1} > \beta_q+1$ we get $c(p_i^j(1)) = \max\{\beta_{q-1}, \beta_q+2\} < \beta_{q-1}+1$, meaning that both arcs leaving $p_i^j(1)$ must be assigned the outcome $(0,\beta_0)$, that is, $p(p_i^j(1)x_i^j(r,2))=0$ for each of $r=1,2$. From this, the capacities of vertices $x_i^j(1,2)$ and $x_i^j(2,2)$ imply $p(x_i^j(r,2)p_i^j(2)) \le q-1$ for both $r=1,2$.

In the general case, a similar argument shows that $p(p_i^j(s-1)x_i^j(r,s))=0$ and $p(x_i^j(r,s)p_i^j(s)) \le q-1$ for each $r=1,\dots,s$ and $s=2,\dots,s_i-1$. The key fact used in the reasoning is $c(p_i^j(s)) < s\beta_{q-1} + 1$, which follows from our assumptions. Thus, for $p_i^j(s_i - 1)$ we obtain that all of its s_i outgoing arcs must be assigned the outcome $(0, \beta_0)$, which in turn implies that $p(b_j\ell') \geq 1$ for each $\ell' \in {L'}_i^j$. Hence, b_j gains at least s_i points resulting from these arcs. From the validity of p we know that $\sum_{i:\varphi(i)=j} s_i \leq c(b_j) = b$, which proves that putting the item of size s_i into the bin $\varphi(i)$ for each $i = 1, \ldots, n$ yields a solution for our UNARY BIN PACKING instance.

Case B, defined precisely as in the proof of Theorem 8, can be handled by combining the ideas used in the previous case and in Case B of the proof of Theorem 8. Instead of giving a formal description, we only give a sketch of the reduction and leave all details to the readers. We start from the graph $H - L'$, where H is defined as in the proof of Theorem 8. First, for each i and j, we replace the subgraphs T_i^j with the graphs P_i^j, adding an arc from $p_i^j(1)$ to ${a'}_i^j$ and connecting $p_i^j(s_i - 1)$ to b_j with s_i parallel arcs. We orient the edges incident to vertices of A such that they become sources. Finally, we replace each arc not incident to A' with three new vertices and four arcs exactly as we did in Case B of Theorem 8. Defining capacities is straightforward, except for the vertices $p_i^j(s)$ which need to receive the capacity value $\max\{(s+1)\beta_{q-1}, s + (s+1)\beta_q\}$. The argument showing the correctness of the reduction is again a straightforward adaptation of the previously used reasoning.

Case C can be handled exactly as in the proof of Theorem 8. □

Theorem 10 and the fact that the treewidth of a graph is always at most its pathwidth imply the following corollary.

Corollary 3 *Except for the case where S is of the form $\{(i, k-i) \mid 0 \leq i \leq k\}$ for some $k \in \mathbb{N}$, ALC(S) is W[1]-hard when parameterized by the treewidth of $\bar{G}$ where G is the input graph.*

We remark that Corollary 3 also follows from Theorem 8, as a graph admitting a feedback vertex set of size k has treewidth at most $k + 1$.

7 Conclusion

The purpose of this paper was to investigate the computational complexity of the sports elimination problem in a detailed, multivariate fashion. To this end, we reformulated the problem into a natural graph labelling problem that we called ARC LABELLING WITH CAPACITIES, and examined it using the tools of both classical and parameterized complexity. Instead of focusing on the set of possible outcomes, we considered various properties of the input instance and their impact on the tractability of the problem; see Table 1 for an overview of the obtained results.

Finding further conditions that make the problem tractable could be the subject of future research. In particular, an intriguing question left open is whether ARC LABELLING WITH CAPACITIES can be solved in FPT time, when the parameter is $\text{fes}((\bar{G})^1)$, the minimum size of a feedback edge set of the undirected

Condition on G	Parameter	Complexity	Reference
forest	-	P	Thm. 2, Rem. 2
cycle	-	P	Thm. 3
max degree 2	-	P	Thm. 4
max degree 3	# of vertices of degree 3	FPT	Thm. 5
-	# of vertices of degree ≥ 3	FPT	Thm. 6
-	feedback edge number of $\bar{G}$	FPT	Thm. 7
-	feedback vertex number of $\bar{G}$	W[1]-hard	Thm. 8
-	treewidth of $\bar{G}$	XP	Thm. 9
-	pathwidth of $\bar{G}$	W[1]-hard	Thm. 10

Table 1. Overview of our results on the complexity of ALC(S) for input graph G. The W[1]-hardness results assume that S, after normalization, is not of the form $\{(i, k-i) \mid i = 0, \ldots, k\}$ for any integer k.

and simple version of the input graph G. The case of bounded capacities might also be worth studying.

Another possible direction for further research is to investigate the following generalization of the sports elimination problem: given a set of n teams with certain current scores and a set of remaining matches to be played between them, is there a way to finish the remaining matches so that our favorite team finishes *not worse than at the r-th place*? In other words, can our team beat at least $n - r$ teams?

References

1. Ilan Adler, Alan L. Erera, Dorit S. Hochbaum, and Eli V. Olinick. Baseball, optimization and the world wide web. *Interfaces*, 32(2):12–22, 2002.
2. Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discret. Math.*, 12(3):289–297, 1999.
3. Ann Becker and Dan Geiger. Approximation algorithms for the loop cutset problem. In *Proceedings of the 10th conference on Uncertainty in Artificial Intelligence*, pages 60–68, San Francisco, CA, 1994. Morgan Kaufmann.
4. Thorsten Bernholt, Alexander Gülich, Thomas Hofmeister, and Niels Schmitt. Football elimination is hard to decide under the 3-point-rule. In *MFCS 1999: Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science*, volume 1672 of *Lecture Notes in Computer Science*, pages 410–418. Springer-Verlag, 1999.
5. Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors. *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*. Springer, 2012.
6. Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth.

In *FOCS 2013: Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 499–508, 2013.

7. Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set new measure and new structures. In *SWAT 2010: Proceedings of the 12th Scandinavian Symposium and Workshops on Algorithm Theory*, volume 6139 of *Lecture Notes in Computer Science*, pages 93–104. Springer, 2010.
8. Rod G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, London, 2013.
9. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, 1979. A Series of Books in the Mathematical Sciences.
10. Dan Gusfield and Charles U. Martel. A fast algorithm for the generalized parametric minimum cut problem and applications. *Algorithmica*, 7(5&6):499–519, 1992.
11. Dan Gusfield and Charles U. Martel. The structure and complexity of sports elimination numbers. *Algorithmica*, 32(1):73–86, 2002.
12. Alan J. Hoffman and Theodore J. Rivlin. When is a team "mathematically" eliminated? In *Proceedings of the Princeton Symposium on Mathematical Programming*, pages 391–401. Princeton University Press, 1970.
13. Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79:39–49, 2013.
14. Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12:415–440, 1987.
15. Walter Kern and Daniël Paulusma. The new fifa rules are hard: complexity aspects of sports competitions. *Discrete Applied Mathematics*, 108(3):317–323, 2001.
16. Walter Kern and Daniël Paulusma. The computational complexity of the elimination problem in generalized sports competitions. *Discrete Optimization*, 1(2):205–214, 2004.
17. Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.
18. Christian Komusiewicz and Rolf Niedermeier. New races in parameterized algorithmics. In *MFCS 2012: Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science*, volume 7464 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 2012.
19. Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.
20. S. Thomas McCormick. Fast algorithms for parametric scheduling come from extensions to parametric maximum flow. *Operations Research*, 47(5):744–756, 1999.
21. Rolf Niedermeier. Reflections on multivariate algorithmics and problem parameterization. In *STACS 2010: Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science*, pages 17–32, 2010.
22. Lawrence W. Robinson. Baseball playoff eliminations: An application of linear programming. *Operations Research Letters*, 10(2):67 – 74, 1991.
23. Benjamin L. Schwartz. Possible winners in partially completed tournaments. *SIAM Review*, 8:302–308, 1966.
24. Kevin D. Wayne. A new property and a faster algorithm for baseball elimination. *SIAM Journal on Discrete Mathematics*, 14(2):223–229, 2001.