Weighted Total Acquisition

Guillaume Bagan, Valentin Gledel, Marc Heinrich, Fionn Mc Inerney

Weighted Total Acquisition*

Guillaume Bagan[1], Valentin Gledel[2], Marc Heinrich[3], and Fionn Mc Inerney[4]

[1]LIRIS, Université Claude Bernard, Lyon, France

[2]G-SCOP, CNRS, Université Grenoble-Alpes, Grenoble, France

[3]School of Computing, University of Leeds, Leeds, United Kingdom

[4]Laboratoire d'Informatique et Systèmes, Aix-Marseille Université, CNRS, and Université de Toulon Faculté des Sciences de Luminy, Marseille, France

Abstract

In the Weighted Total Acquisition problem (WTA) on a weighted graph G (only positive weights), a vertex v can acquire the total weight of a neighbour u if and only if the current weight of v is at least that of u. The new weight of u is then zero, and the new weight of v is then the sum of the weights at u and v just before the acquisition. Over all possible acquisition sequences in G with weight function w, the minimum number of vertices with a non-zero weight at the end is denoted by $a_t(G, w)$. Given a graph G, a weighting w, and an integer $k \geq 1$, the WTA problem asks whether $a_t(G, w) \leq k$. The Binary (Unary resp.) WTA problem corresponds to the WTA problem when the weights are encoded in binary (unary resp.).

We prove that the Binary WTA problem is NP-complete in trees of bounded degree, trees of bounded depth, and wheels, but that it is in XP for trees and wheels when parameterized by the solution size. Moreover, we show that Binary WTA is NP-complete in $K_{3,n}$, planar graphs of pathwidth 2, and unit interval graphs even when $k = 1$, and in trivially perfect graphs when $k \geq 2$ (but polynomial-time solvable when $k = 1$).

We prove that Unary WTA is polynomial-time solvable in graphs of bounded treewidth and degree. When only the treewidth is bounded, this algorithm is quasi-polynomial, *i.e.*, it runs in time $W^{O(\log W)}$, where W is the sum of the weights of the vertices. Moreover, we show that Unary WTA is FPT in trees when parameterized by the maximum degree. However, we show that WTA is strongly NP-complete in trivially perfect graphs and split graphs, even when $k = 1$ in the latter.

Keywords: Total acquisition, Complexity, Treewidth, Dynamic programming.

*This work has been partially supported by ANR projects GAG and DISTANCIA (ANR-17-CE40-0015 and ANR-14-CE25-0006, respectively).

1 Introduction

Data gathering is an essential capability of any type of wireless network. In general, all of the nodes of the network want to send their data to a particular node, typically known as a sink. In this process, nodes may act as intermediary sinks, in that, they collect data from many of the other nodes, and then, the intermediary sinks transfer their data to the destination or final sink. This process is known as multi-hopping. This all-to-one type of communication or data gathering protocol is called convergecast. Convergecast has been vastly studied and has found its applications in, *e.g.*, wireless sensor networks (WSN) [6, 18, 19], wireless body area networks (WBAN) [1, 5], wireless ad hoc radio networks [13], and vehicular networks [10].

In 1995, Lampert and Slater introduced the following convergecast model. In a graph G, each vertex $v \in V(G)$ initially has a weight of 1, and a vertex x can transfer its total weight $w(x)$ to a neighbouring vertex y if and only if $w(x) \leq w(y)$. Such a transfer is called an *acquisition move*, and when the context is clear, we will simply refer to it as a *move* or *acquisition*. When there are no more possible moves, the remaining set of vertices with non-zero weights is referred to as a *residual set*. The minimum cardinality of a residual set is called the *total acquisition number* of G and denoted by $a_t(G)$ [14]. The rule that a vertex may only transfer its weight to a neighbouring vertex with at least the same weight can be motivated by disjoint set data structures. For example, in the UNION-FIND data structure with linked lists, a smaller list should always be appended to a larger list to improve performance [2].

1.1 Related Work

In 1995, in the seminary paper on the topic, Lampert and Slater proved that $a_t(G) \leq \left\lceil \frac{n+1}{3} \right\rceil$ for any connected graph G on n vertices and that this bound is sharp [14]. The graphs achieving this equality were then characterized in [16]. In 2008, Slater and Wang proved that deciding whether $a_t(G) = 1$ is NP-complete [21]. Further studies on total acquisition have also been done for trees [15], grids [17], and random graphs [2, 9]. Work has also been done on a game variation of total acquisition [20] and variants where the total weight need not be transferred from vertex to vertex, but rather just a portion of it, see, *e.g.*, [11, 22].

In this paper, we consider the *weighted total acquisition number* of a weighted graph G with weight function w which contains only positive integer weights, denoted by $a_t(G, w)$, which is the minimum cardinality of a residual set when the vertices begin with their weight according to w. This is in contrast to the total acquisition number, in which all the vertices begin with weight 1, and makes for a much more realistic model since all the nodes would not necessarily begin with the same amount of data (see, *e.g.*, [6]). To the best of our knowledge, the only paper to have considered this variant is [8], a paper of Godbole et al. from 2017. In [8], they studied the possible sizes of residual sets obtainable for cliques, paths, cycles, wheels, and complete bipartite graphs. Their main focus,

however, was studying $a_t(P_n, w)$ (P_n is a path on n vertices) when w is a random weight function. In particular, they gave bounds on the expected weighted total acquisition number of paths, and, given a weighting w, they described a simple polynomial-time greedy algorithm for finding $a_t(P_n, w)$. The latter algorithm also shows that $a_t(C_n, w)$ (C_n is a cycle on n vertices) can be determined in polynomial time by testing all of the possible $O(n)$ first acquisition moves and then applying the greedy algorithm to the path that remains.

1.2 Our Results

In this paper, we are interested in the complexity of computing the weighted total acquisition number for various classes of graphs. The complexity of the problem depends on whether the weights given in the input are encoded in unary or in binary. Our results include both positive and negative results for both versions of the problem. The paper is organized as follows. We start with some notations and formal definitions in Section 2. We then focus on the binary version of the problem in Section 3. For this version of the problem, most of our results are negative, showing that the problem is hard even for very restricted classes of graphs. In Subsection 3.1, we show that binary weighted total acquisition is NP-complete, even on trees of bounded degree and on trees of bounded depth. On the positive side, we prove that the problem is in XP on trees when parameterized by the weighted total acquisition number of the graph (*i.e.*, $a_t(G, w)$ can be computed in polynomial time if it is upper bounded by a fixed constant). Since the problem is already difficult on trees, our focus for the rest of this section is on graphs which are even more simple, or on the case where the weighted total acquisition number of the graph is bounded. Unfortunately, our results are again mostly negative. We show in Subsection 3.2 that it is NP-complete to decide whether the weighted total acquisition number of $K_{3,n}$ is equal to 1, where $K_{3,n}$ is the complete bipartite graph with one side of the bipartition of size 3. In Subsection 3.3, we prove that deciding whether the weighted total acquisition number of a trivially perfect graph is 1 can be done in polynomial time, while deciding whether this number is at most 2 is NP-complete for this class, as well as for the class of planar graphs with pathwidth at most 2 (studied in Subsection 3.4) and unit-interval graphs. We also show, in Subsection 3.4, that computing the weighted total acquisition number of a wheel (*i.e.*, a cycle plus a universal vertex) is NP-complete, but is also in XP when parameterized by the weighted total acquisition number of the graph.

When the weights on the vertices of the input graph are encoded in unary, the problem appears to be much more tractable. In Subsection 4.1, we show that when the input weights are encoded in unary, there is a polynomial-time algorithm which computes the weighted total acquisition number on graphs with both bounded degree and bounded treewidth. When only the treewidth of the graph is bounded, the algorithm has a quasi-polynomial run time, *i.e.*, it runs in time at most $W^{O(\log W)}$, where W is the sum of the weights of the vertices. In particular, this implies that the problem can be solved in quasi-polynomial time on trees, showing that, unlike the binary version of the problem, it is

unlikely to be NP-hard on trees. Moreover, we show in Subsection 4.2 that the unary version of the problem is FPT when parameterized by the maximum degree of the tree. In other words, the problem can be solved on trees in time $O(f(\Delta)W^c)$, for some computable function f and a constant c, where Δ is the maximum degree of the graph. Finally, in Subsection 4.3, we show that even in its unary version, the problem remains NP-complete on trivially perfect graphs. Moreover, deciding whether the weighted total acquisition number is one on split graphs is NP-complete, even in the unary version of the problem.

2 Notations and Definitions

2.1 Graph theory

We start with some basic definitions and notations from graph theory. A graph $G = (V, E)$ is defined by a *set of vertices* V, and a *set of edges* E. Throughout this paper, every graph that we consider is undirected and simple. Moreover, all our graphs will be given a *weight function* $w : V \to \mathbb{N}$ which assigns weights to the vertices of the graph. Given two vertices u and v of a graph G, uv denotes the edge between u and v, and $N_G(v) = \{u \in V, uv \in E\}$ denotes the *neighbours* of v. If the graph G is clear from the context, then we will drop the subscript and simply write $N(v)$ for the neighbourhood of v. A vertex is *universal* if it is adjacent to all of the vertices of the graph.

Given a graph G with a weight function w, the total weight of the graph G, denoted by W, is the sum of the weights of the vertices, *i.e.*, $W = \sum_{v \in V(G)} w(v)$.

A *tree decomposition* of a graph G is a tree T whose nodes are each labelled with a subset of vertices of the graph G, and such that:

- for every vertex v, the nodes which contain v induce a subtree of T;
- for every edge uv, there is a node of T that contains u and v in its label.

A tree decomposition has width k if the largest subset in the labels of T contains $k+1$ vertices. Trees are exactly the graphs of treewidth 1. The *treewidth*, $\mathrm{tw}(G)$, of a graph G, is the smallest value k such that G admits a tree decomposition of width k. If we restrict T to be a path, then we obtain another parameter called the *pathwidth* of the graph G.

Throughout the paper, we will consider several classes of graphs. Below, we give some formal definitions of these classes. A graph is an *interval graph* if each vertex can be represented by an interval on the real line such that two vertices of the graph are adjacent if and only if their corresponding intervals intersect. *Unit interval graphs* are a subclass of interval graphs, where each vertex is represented by an interval of the same length. Another subclass of interval graphs are the trivially perfect graphs. There are several ways to characterize this class, one way is to say that it has an interval representation such that any two intervals are either disjoint or one is included in the other. Another way to characterize this class is by the following property:

Lemma 1. *A graph is trivially perfect if and only if it can obtained by starting from a disjoint union of isolated vertices (vertices of degree zero), and adding universal vertices.*

Another class of graphs that we will consider are *split graphs*, for which there is a partition of the vertices into an independent set and a clique. Interval graphs and split graphs are a subclass of *chordal graphs*, which are the graphs that do not contain any induced cycle of length four or more. Finally, *planar graphs* are graphs that can be drawn on the plane such that no two edges intersect.

2.2 Problem definition

Let us now give a formal definition of our problem. Given a graph $G = (V, E)$, a weight function w is interpreted as the number of tokens on each of the vertices of G. An *acquisition move* consists in moving $w(v)$ tokens from a certain vertex v to a neighbour u, provided that $w(u) \geq w(v)$. An *acquisition sequence* is a sequence of weight functions $w_1, \ldots, w_t$ such that w_{i+1} is obtained from w_i by an acquisition move. Given an acquisition sequence, its *residual set* is the set of vertices which have at least one token at the end of the sequence, *i.e.*, it is the set $\{v \in V, w_t(v) > 0\}$. The cost of an acquisition sequence is the size of the residual set. Our problem consists in finding an acquisition sequence of minimum cost, *i.e.*, which minimizes the size of the residual set. Formally, it has the following definition:

Weighted Total Acquisition problem (WTA) :

Input: A graph G with a weight function w, and an integer $k \geq 1$.
Question: Is there an acquisition sequence of cost at most k?

We will denote by $a_t(G, w)$ the (weighted) total acquisition number of G, which is the minimum cost of an acquisition sequence for the graph G with weight function w. Hence, the problem above consists in deciding whether $a_t(G, w) \leq k$ or not. Note that the WTA problem is clearly in NP, and so this fact will be omitted from all of the (strongly) NP-completeness proofs.

Throughout this paper, we study the complexity of this problem, and, specifically, when the graph G is restricted to various classes of graphs. From a complexity point of view, it is important to specify whether the weights in the weight function given as input are encoded in unary or in binary. Since we have results for both versions of the problem, we will explicitly write Unary WTA (resp. Binary WTA) when the input weights are encoded in unary (resp. binary). Some of our hardness results also apply to the case where the parameter k, the target cost for the acquisition sequence, is not given as input, but is instead a fixed constant. This variant of the problem is denoted by k-WTA.

A first observation concerning this problem is that there are several ways to encode a solution to the problem. The trivial way to represent a solution consists in simply giving the sequence of acquisition moves. However, there are other more simple ways to represent a solution, and the sequence of moves can also be recovered from these other representations.

The first alternative way to define a solution to the problem consists in giving, for every edge of the graph, how much weight is transferred along this edge (note that an edge can be used at most once in any acquisition sequence). Hence, a solution can be represented by a weight function f on the edges of the graph G. In the rest of the paper, when we use such a weight function f, we assume that the edges of G are oriented. More precisely, given two vertices u and v, $f(uv)$ represents the number of tokens which are moved from u to v. This quantity can be negative if the tokens are moved from v to u, and, in particular, $f(uv) = -f(vu)$. To avoid confusion, when considering a weight function on an edge, we will always specify the two endpoints unless it is clear from the context. We will also abuse notations, and sometimes write $f(uv)$ for two vertices which are not adjacent, in which case the value is always zero.

Moreover, to simplify some notations, given a weight function f on the edges of G, we will denote by f^+ the function such that $f^+(uv) = \max(0, f(uv))$. Consequently, f^+ allows to count the number of tokens which are moved in a particular direction. Using this notation, for a given vertex v, the quantity $\sum_{u \in N(v)} f^+(uv)$ denotes the total number of tokens received by the vertex v from its neighbours. Note that not all weight functions correspond to valid acquisition sequences. A weight function f is a valid solution if there is an acquisition sequence where the number of tokens which are moved on a given edge is exactly the value of f on this edge. It is easy to check that a weight function f is a valid solution if and only if it satisfies the following local constraints for every vertex v of the graph:

(P1) v gives tokens to at most one of its neighbours, *i.e.*, there is at most one $u \in N(v)$ with $f(vu) > 0$;

(P2) either v does not give tokens, or it gives as many as it received plus its own. In the latter case, we must have:

$$w(v) + \sum_{x \in N(v)} f(xv) = 0;$$

(P3) v must be able to acquire all the incoming weights. In other words, there is an ordering $u_1, \ldots, u_k$ of the neighbours of v, such that :

$$w(v) + \sum_{j<i} f^+(u_j v) \geq f^+(u_i v).$$

Since the constraints that a solution f must satisfy are local, this representation is convenient for dynamic programming algorithms in the unary version of the problem. Note that specifying the number of tokens which are moved on the edges of the graph in the acquisition sequence is not strictly necessary, and it is sufficient to store which edges are used in the sequence (*i.e.*, have a non-zero weight). This leads to the last way to encode a solution to the problem.

An *acquisition forest* is a rooted spanning sub-forest F (*i.e.*, all the trees in the forest are rooted) of the graph G such that every connected component

of the forest has a weighted total acquisition number of 1. The root of each tree in the forest corresponds to the vertex in the component with remaining tokens at the end of the acquisition sequence, and the edges of the tree, oriented towards the root, tell us in which directions the tokens are moved along the edges. Consequently, with this representation, the cost of the acquisition forest is simply the number of connected components. It is easy to see that the three ways to represent a solution are equivalent: if we have only an acquisition forest, getting an acquisition sequence with the same cost can be done in polynomial time. Hence, when we devise algorithms to solve the WTA problem, the algorithm will compute the solution in one of the three forms, whichever is more convenient at the time.

2.3 Hardness reductions

All of our hardness results in the binary version are achieved by using reductions from the PARTITION problem, which asks whether a set of integers can be partitioned into two subsets of equal value:

PARTITION problem :

Input: A set $S = \{x_1, \ldots, x_\ell\}$ with $M = \sum_{1 \leq i \leq \ell} x_i$.

Question: Is there a partition (S_1, S_2) of S such that $\sum_{y \in S_1} y = \sum_{z \in S_2} z = \frac{M}{2}$?

This problem is a special case of another well-known problem called subset sum and is a known NP-hard problem when the values x_i in the input are encoded in binary [12]. However, if the values are written in unary, it can be solved using a simple dynamic programming algorithm [3]. Hence, this problem is not suitable in reductions which aim at proving that UNARY WTA is NP-hard. In this case, we will use the following problem instead:

3-PARTITION problem :

Input: A set $S = \{x_1, \ldots, x_{3n}\}$ and an integer B such that $\sum_{1 \leq i \leq 3n} x_i = n \cdot B$, and $\frac{B}{4} < x_i < \frac{B}{2}$ for all $1 \leq i \leq 3n$.

Question: Is there a partition of S into n triples $S_1, \ldots, S_n$ such that $\sum_{x \in S_i} x = B$ for all $1 \leq i \leq n$?

This problem was shown to be strongly NP-hard in [7], *i.e.*, the hardness holds even if the input is encoded in unary.

3 Binary WTA

We start by considering the complexity of the binary version of the problem, and we study its complexity for several classes of graphs. Not so surprisingly, the problem turns out to be difficult, and is NP-complete even for very simple classes

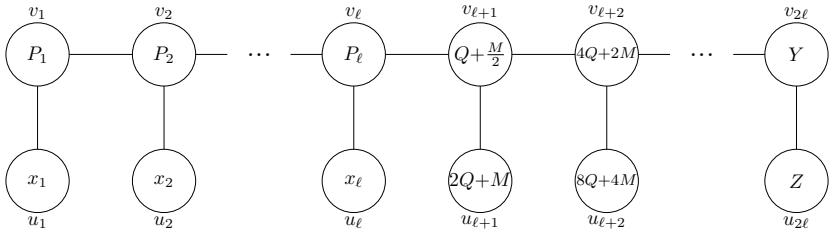

Figure 1: Construction of the caterpillar graph G from the proof of Theorem 1, where $Y = 4^{\ell-1}(Q + \frac{M}{2})$ and $Z = 4^{\ell-1}(2Q + M)$.

of graphs such as trees or wheels. However, we prove the problem is tractable on trees when the size of the residual set is bounded, but we show that for other classes of graphs such as complete bipartite graphs, it remains NP-complete to even decide if the weighted total acquisition number of the graph is equal to one. For trivially perfect graphs, Binary 1-WTA is polynomial-time solvable, but Binary 2-WTA is NP-complete. These results prove that polynomial-time algorithms are unlikely to exist for all but the simplest classes of graphs such as paths and cycles, for which such algorithms exist (see Subsection 1.1 and [8]).

3.1 Trees

We start by considering the complexity of the problem on trees. In this section, we show that the problem is NP-complete, even for caterpillars of maximum degree 3 (Theorem 1) and subdivided stars (Theorem 2). This shows that neither bounded degree nor bounded depth are sufficient conditions to make the problem polynomial-time solvable.

Theorem 1. *The* Binary WTA *problem is NP-complete on caterpillars of maximum degree 3.*

Proof. The NP-hardness proof is done by a reduction from the Partition problem. Let $S = \{x_1, \ldots, x_\ell\}$ be an instance for the Partition problem. We construct, in polynomial time, an instance G with a weight function w for the WTA problem. This graph G will be such that $a_t(G, w) \leq \ell$ if and only if the set $S = \{x_1, \ldots, x_\ell\}$ can be partitioned into two subsets S_1 and S_2 of equal value.

The construction of $G = (V, E)$ is as follows. The vertices of G will be $V = \{v_1, \ldots, v_{2\ell}\} \cup \{u_1, \ldots, u_{2\ell}\}$, where the vertices v_i induce a path, and the u_i form an independent set with u_i adjacent only to v_i for every $1 \leq i \leq 2\ell$. See Figure 1 for an illustration of the construction.

Let $M = \sum_{i \leq \ell} x_i$, and let $P_1, \ldots, P_\ell, Q$ be integers such that $P_1 > M$, $P_j > P_{j-1} + M$ for all $2 \leq j \leq \ell$, and $Q = P_1 + P_2 + \ldots + P_\ell$. The weight function

on G is taken such that for every $1 \leq i \leq \ell$:

$$w(v_i) = P_i \qquad w(v_{\ell+i}) = 4^{i-1}\left(Q + \frac{M}{2}\right)$$
$$w(u_i) = x_i \qquad w(u_{\ell+i}) = 4^{i-1}\left(2Q + M\right)$$

To complete the proof, we only need to show that $a_t(G, w) \leq \ell$ if and only if S has a partition into two subsets of equal value. We start with the forward direction and let us assume that $a_t(G, w) \leq \ell$. We want to prove that there is a subset S_1 of S such that $\sum_{x \in S_1} x = \frac{M}{2}$. Let us consider an optimal acquisition sequence for G. Since the residual set of this sequence has size at most ℓ, there is at least one index i such that the tokens from $u_{\ell+i}$ are moved towards its (only) neighbour $v_{\ell+i}$. However, since $w(v_{\ell+i}) = 4^{i-1}(Q + \frac{M}{2})$ is smaller than $w(u_{\ell+i}) = 4^{i-1}(2Q + M)$, this means that $v_{\ell+i}$ must first receive tokens from one of its two other neighbours. Since $v_{\ell+i+1}$ has a larger weight than $v_{\ell+i}$, it cannot receive tokens from this neighbour, and consequently it must receive tokens from $v_{\ell+i-1}$. The number of tokens it has to receive is at least $w(u_{\ell+i}) - w(v_{\ell+i}) = 4^{i-1}(Q + \frac{M}{2})$. Moreover, this quantity is four times the initial weight of $v_{\ell+i-1}$, hence, the vertex $v_{\ell+i-1}$ itself must receive tokens from both of its other neighbours. By iterating the argument, it follows that the tokens from $u_{\ell+1}$ are moved to its neighbour $v_{\ell+1}$, which then transfers these tokens to its neighbour on the right. This operation is possible if and only if it receives exactly $Q + \frac{M}{2}$ tokens from its neighbour v_ℓ (it cannot receive more than that since a vertex cannot receive more than its current weight in one move). If we denote by I the set of indices $1 \leq i \leq \ell$ such that u_i transfers tokens to v_i in the acquisition sequence, then the previous condition implies that $\sum_{i \in I} x_i = \frac{M}{2}$.

Let us now prove the other direction, and assume that the set S can be partitioned into two subsets S_1 and S_2 such that $\sum_{y \in S_1} y = \sum_{z \in S_2} z = \frac{M}{2}$. We want to prove that $a_t(G, w) \leq \ell$. Using the partition, for every $1 \leq i \leq \ell$, if $x_i \in S_1$, then v_i acquires its neighbour u_i. Then, for each $2 \leq i \leq \ell$, v_i acquires v_{i-1}. At this point, the vertex v_ℓ has a total weight of $\sum_{1 \leq i \leq \ell} P_i + \sum_{x \in S_1} x = Q + \frac{M}{2}$. From this point, for $1 \leq j \leq \ell$, $v_{\ell+j}$ first acquires $v_{\ell+j-1}$, and then, acquires $u_{\ell+j}$. Checking that all these moves are valid is straightforward. The residual set has size at most ℓ ($v_{2\ell}$ plus at most $\ell - 1$ of the vertices $u_1, \ldots, u_\ell$). □

The previous result shows that the problem is hard if we allow long paths in a tree, even if the maximum degree is 3. The following theorem compliments this result by showing that even if the depth of the tree is bounded (and the maximum degree is unbounded), the problem remains NP-complete. Note that trees with both bounded depth and bounded maximum degree have bounded size and consequently are not interesting to study.

Theorem 2. *The* BINARY WTA *problem is NP-complete on stars subdivided once.*

Proof. The NP-hardness of the problem is shown by a reduction from the Partition problem. Let $S = \{x_1, \ldots, x_\ell\}$ be an instance for Partition. We construct, in polynomial time, a graph G with a weight function w such that $a_t(G, w) \leq \ell + 1$ if and only if the set $S = \{x_1, \ldots, x_\ell\}$ can be partitioned into two subsets S_1 and S_2 of equal value.

The graph G is a star where each edge is subdivided once. We denote by c the center of the star, $u_1, \ldots u_\ell, t_1, t_2$ the leaves of the star, and $v_1, \ldots, v_\ell, s_1, s_2$ the vertices obtained by the subdivision of the edges. See Figure 2 for an illustration of the construction.

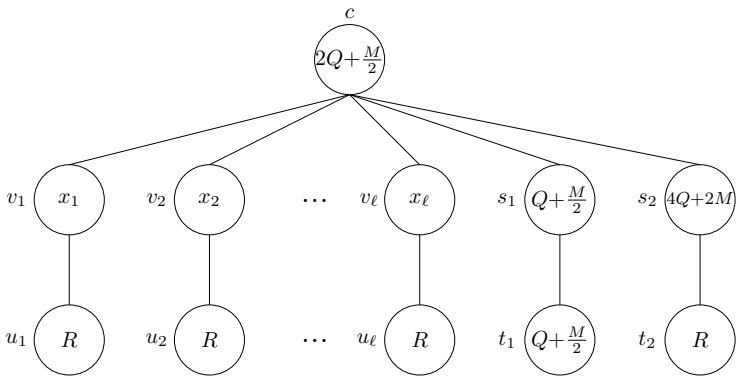

Figure 2: Construction of the subdivided star graph G from the proof of Theorem 2.

Let $M = \sum_{1 \leq i \leq \ell} x_i$, and let Q, R be integers such that $Q > M$ and $R > 15Q$. The weight function w is defined such that for every $1 \leq i \leq \ell$:

$$\begin{aligned} w(v_i) &= x_i & w(u_i) &= R \\ w(s_1) &= Q + \frac{M}{2} & w(t_1) &= Q + \frac{M}{2} \\ w(s_2) &= 4Q + 2M & w(t_2) &= R \\ w(c) &= 2Q + \frac{M}{2} \end{aligned}$$

First, assume that $a_t(G, w) \leq \ell+1$, and let us prove that S admits a partition into two subsets of equal value. Note that, since the vertices $u_1, \ldots, u_\ell, t_2$ form an independent set and all these vertices have sufficiently large weight, they will never be acquired by any other vertex. Hence, these vertices must form the residual set of any optimal acquisition sequence. This implies that in such a sequence, t_1 must be acquired by s_1, and c must be acquired by s_2 since none of the other vertices v_i has an initial weight sufficiently large to acquire c. The vertex c can acquire s_1 if and only if it reaches a weight of at least $w(s_1) + w(t_1)$ since s_1 acquired t_1. This implies that c must acquire a weight of at least $\frac{M}{2}$ from the vertices $v_1, \ldots, v_\ell$. Moreover, since c is acquired by s_2, its weight at

the moment it is acquired cannot exceed $w(s_2) = 4Q + 2M$. This implies that c did not acquire more than $\frac{M}{2}$ from the vertices $v_1, \ldots, v_\ell$. Hence, if S_1 is the set of weights acquired by c, then $(S_1, S \setminus S_1)$ is a partition of S where the two parts sum to the same total value.

For the other direction, assume that S can be partitioned into two subsets S_1 and S_2 such that $\sum_{y \in S_1} y = \sum_{z \in S_2} z = \frac{M}{2}$. Consider the following acquisition sequence:

1. for all $1 \leq i \leq \ell$, if $x_i \in S_1$, then the vertex v_i is acquired by c, otherwise, it is acquired by u_i;
2. s_1 acquires t_1;
3. at this step, c has a total weight of $w(c) + \frac{M}{2}$ which is equal to the current weight of s_1, hence, c can acquire s_1 and its new weight becomes $4Q+2M$;
4. finally, c can be acquired by s_2 which can then be acquired by t_2.

The residual set of this acquisition sequence has size exactly $\ell + 1$. □

The two previous results showed that Binary WTA is hard, even in trees. On the positive side, the theorem below shows that if the weighted total acquisition number is bounded by a fixed constant, then the problem becomes polynomial-time solvable. From a parameterized complexity point of view, this result implies that Binary k-WTA is in XP in trees when parameterized by k. The question of whether this problem is FPT or not is still open.

Theorem 3. *For all integers $k \geq 1$, the* Binary k-WTA *problem can be solved in trees in time $n^{O(k)}$.*

Proof. The result follows from the two following observations:

- The first observation is that there are at most $O(n^k)$ ways to partition a tree into k connected components. Indeed, every edge which is removed from the tree splits a component into 2, thus a partition of size k is uniquely defined by the k edges which were removed from G.
- We can decide in polynomial time if a tree has a weighted total acquisition number of 1. For this, simply observe that the first move of an acquisition sequence on the tree must move tokens from a leaf to its neighbour (since otherwise the move disconnects the graph, and the residual set will have size at least 2). Moreover, the order according to which the leaves are acquired is not important.

With these two observations, devising an algorithm for the problem is straightforward: we can simply try all $O(n^k)$ partitions, and for each of these partitions we can check in polynomial time if the weighted total acquisition number of each connected component is one. □

3.2 Complete bipartite graphs

In the previous section, we studied the complexity of computing the weighted total acquisition number on trees, and saw that the problem was NP-complete if

the weighted total acquisition number of the tree is not bounded. In this section, we show that it remains NP-complete to decide if the weighted total acquisition number is equal to 1 for another very simple class of graphs: complete bipartite graphs. In Theorem 4, we show that this problem is NP-complete in $K_{3,n}$, the complete bipartite graph where one side of the bipartition has size 3.

Theorem 4. *The* Binary 1-WTA *problem is NP-complete in complete bipartite graphs, even if one side has size* 3*.*

Proof. To show that the problem is NP-hard, we reduce from the Partition problem. Given an instance $S = \{x_1, \ldots, x_\ell\}$ for the Partition problem, we construct, in polynomial time, a graph G with a weight function w such that $a_t(G, w) = 1$ if and only if the set S can be partitioned into two subsets S_1 and S_2 of equal value. The graph G is $K_{3,\ell+1}$, with v_1, v_2, v_3 the vertices of the side of size 3, and $u_1, \ldots, u_{\ell+1}$ the vertices of the other side. Let $M = \sum_{x \in S} x$, and Q, R be integers such that $Q > M$, and $R > 10Q$. The weight function w is constructed such that $w(v_1) = R$, $w(v_2) = 2Q + M$, $w(v_3) = Q$, $w(u_{\ell+1}) = Q + \frac{M}{2}$, and for every $1 \leq i \leq \ell$, take $w(u_i) = x_i$. See Figure 3 for an illustration of the construction.

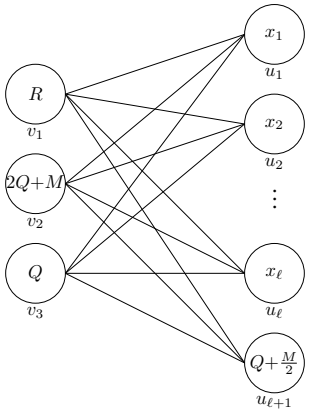

Figure 3: Construction of the complete bipartite graph G from the proof of Theorem 4.

First, assume that $a_t(G, w) = 1$, and let us prove that there is a partition (S_1, S_2) of S into subsets of equal value. Consider an optimal acquisition sequence for the graph G. Since R is taken sufficiently large and $a_t(G, w) = 1$, the residual set must be equal to $\{v_1\}$. This implies that v_2 must be acquired by one of its neighbours, and this neighbour must be $u_{\ell+1}$ since it is the only vertex in its neighbourhood that can reach a sufficient weight to acquire v_2. In order to acquire v_2, $u_{\ell+1}$ must first reach a sufficient weight to acquire it, hence, it must first acquire at least $Q + \frac{M}{2}$ tokens from v_3. Moreover, since the initial weight of $u_{\ell+1}$ is also $Q + \frac{M}{2}$, it must acquire exactly $Q + \frac{M}{2}$ tokens

from v_3 (since otherwise the move from v_3 to $u_{\ell+1}$ would not be possible). This implies that v_3 must acquire a weight of exactly $\frac{M}{2}$ from the vertices $u_1, \ldots, u_\ell$. Consequently, if S_1 corresponds to the weights of the vertices acquired by v_3 in this acquisition sequence, and $S_2 = S \setminus S_1$, then (S_1, S_2) forms a partition of S into subsets of equal value.

Now, let us prove the other direction, that is, if the set S can be partitioned into two subsets S_1 and S_2 such that $\sum_{y \in S_1} y = \sum_{z \in S_2} z = \frac{M}{2}$, then $a_t(G, w) \leq 1$. Let S_1 and S_2 be such a partition of S. We start by using this partition to send a total weight of $\frac{M}{2}$ from the vertices $u_1, \ldots, u_\ell$ to v_3 by making v_3 acquire all the weights x_i, for $x_i \in S_1$ for example. By this operation, v_3 reaches a weight of $Q + \frac{M}{2}$, and can be acquired by $u_{\ell+1}$ who now reaches a weight of $2Q + M$, and can thus acquire v_2. Finally, v_1 can acquire all the remaining vertices, which shows that $a_t(G, w) = 1$. □

Note that the proof can be easily adapted to k-WTA, for $k > 1$ by adding vertices with very large weights on the side of size 3. In this case, the graph we obtain with this construction will be $K_{k+2,\ell+1}$.

3.3 Trivially perfect graphs

In this section, we focus on the weighted total acquisition number of trivially perfect graphs. We give a full dichotomy of the complexity of the problem in terms of the maximum size of the residual set. More precisely, we prove that Binary 1-WTA can be solved in polynomial time on trivially perfect graphs, while Binary 2-WTA is NP-complete for this class.

Theorem 5. *The* Binary 1-WTA *problem is polynomial-time solvable on trivially perfect graphs.*

The algorithm in this theorem will find an acquisition sequence with a residual set of size 1 (if it exists) by greedily moving tokens from one vertex to one of the universal vertices of the graph. The correctness of the algorithm relies on the following lemma:

Lemma 2. *Let $G = (V, E)$ be a graph and w be a weight function of G. Assume that there is a subset S such that:*

- *$G[V \setminus S]$ contains k connected components $A_1, \ldots, A_k$;*
- *$\sum_{v \in S} w(v) < w(u)$ for every $u \in V \setminus S$.*

Then, $a_t(G, w) \geq k$.

Proof. Assume by contradiction that $a_t(G, w) < k$, and let us consider an optimal acquisition sequence. Since $a_t(G, w) < k$, this implies that at some point during the acquisition sequence, one of the vertices in one of the sets A_i is acquired by a vertex which is not in A_i, and thus, is in S. Let us consider the first time such an event happens, and let v be the vertex in A_i, and u the vertex in S. When v is acquired by u, there are at least $w(v)$ tokens on v, while the number

of tokens on u can be at most $\sum_{x \in S} w(x)$, since this is the first time that a vertex in S acquires a vertex outside S. However, this contradicts the assumption that $\sum_{x \in S} w(x) < w(v)$, and consequently is not possible. Hence, $a_t(G, w) \geq k$. □

The algorithm for finding an acquisition sequence proceeds greedily. At each step, it finds a universal vertex with maximum weight, and makes it acquire another vertex of the graph until only it remains. If the algorithm cannot find such an acquisition move, then it concludes that the weighted total acquisition number of the graph is larger than 1. The algorithm is described more formally below:

Input: A connected trivially perfect graph G and a weight function w.
Output: Tests whether $a_t(G, w) = 1$.

```
while |V(G)| > 1 do
    u ← universal vertex of G with maximum weight.
    if there is v ≠ u such that w(v) ≤ w(u) then
        u acquires v
        remove v from G
    else
        return False
    end if
end while
return True
```

Proof of Theorem 5. Clearly, the algorithm described above runs in polynomial time since the number of vertices in the graph decreases by one at each iteration. Hence, to prove the theorem, we only need to prove that it is correct. The case when the algorithm returns true is straightforward since the algorithm has found an acquisition sequence with only one vertex at the end. Hence, we only need to prove that when the algorithm returns false, then $a_t(G, w) > 1$. This is done by showing that when the algorithm returns false, then we can construct a set S which satisfies the conditions of Lemma 2.

Let us assume that the algorithm returned false on the graph G with the weight function w, and let us show that $a_t(G, w) > 1$. Let G' and w' be the graph and the weight function at the time where the algorithm returned false. Let us start with a few observations. First, remark that G' contains exactly one universal vertex u. Indeed, if there were two universal vertices, then the one with the larger weight could acquire the one with the smaller weight. Also, since the graph remains connected throughout the algorithm, it must contain at least one universal vertex by the definition of trivially perfect graphs.

Furthermore, at each step, the universal vertex u acquires another vertex of the graph, hence, if S is the set of vertices which were removed up to this point, then we must have $w'(u) = w(u) + \sum_{v \in S} w(v)$ since u is the only universal vertex in G'. Consequently, we also have $w'(v) = w(v)$ for all $v \in G' \setminus \{u\}$.

Since u is the unique universal vertex of G', then $G' \setminus \{u\}$ is not connected. Since the algorithm returned false at this step, then for all $v \in G'$ with $v \neq u$, we

have $w'(v) > w'(u) = \sum_{x \in S \cup \{u\}} w(x)$. Hence, $S \cup \{u\}$ is a separator of G, and by the previous inequality and Lemma 2, we can conclude that $a_t(G, w) \geq 2$. □

The Binary WTA problem becomes NP-complete as soon as the desired residual set is of size at most k for $k \geq 2$.

Theorem 6. *The* Binary 2-WTA *problem is NP-complete on trivially perfect graphs.*

Proof. To show that the problem is NP-hard, we reduce from the Partition problem. Given an instance $S = \{x_1, \ldots, x_\ell\}$ for Partition, we construct, in polynomial time, a graph G and a weight function w such that $a_t(G, w) \leq 2$ if and only if S can be partitioned into two subsets S_1 and S_2 of equal value.

The graph G is composed of two independent sets $I_v = \{v_1, v_2\}$ and $I_u = \{u_1, \ldots, u\ell, s_1, s_2, t_1, t_2\}$, and a universal vertex c. The vertex v_1 is only adjacent to t_1, t_2, and c, while v_2 is adjacent to $u_1, \ldots, u_\ell, s_1, s_2$, and c.

Let $M = \sum_{1 \leq i \leq \ell} x_i$, and let P, Q be integers such that $P > \frac{M}{2}$ and $Q > P + M$. The weight function w is as follows for all $1 \leq i \leq \ell$ and $1 \leq j \leq 2$:

$$w(c) = P \qquad w(v_j) = Q \qquad w(u_i) = x_i$$
$$w(s_j) = Q + \frac{M}{2} \qquad w(t_j) = Q + P + \frac{M}{2}$$

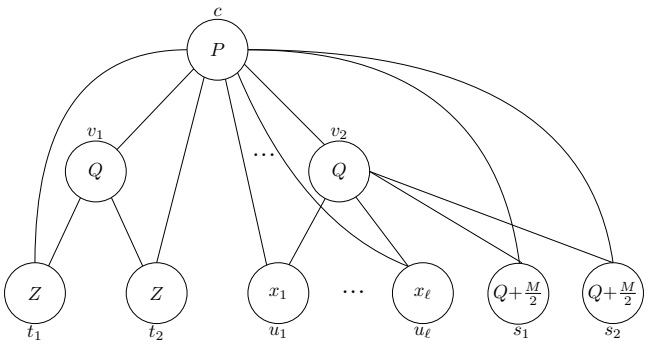

Figure 4: Construction of the trivially perfect graph G from the proof of Theorem 6, where $Z = Q + P + \frac{M}{2}$.

The graph G is trivially perfect, as we can see in Figure 4. We start with the following claim:

Claim 1. *In any acquisition sequence of G, c never acquires any vertex in $S^* = \{v_1, v_2, t_1, t_2, s_1, s_2\}$.*

Proof of claim. Assume by contradiction that there is an acquisition sequence of G in which c acquires a vertex of S^*, and let us be at the step in the acquisition sequence just before the first time such an acquisition occurs. At this time, the current weight of c is at most $P+M$ since c has not acquired any of the vertices in S^* yet. Moreover, all the vertices in S^* have an initial weight of at least Q, hence, their current weight, if it is not zero, is also at least Q. But, $Q > P+M$, which means that the acquisition of the vertex in S^* by c is not valid. ◇

First, assume that $a_t(G,w) \leq 2$ and let us prove that there is a partition of S into two subsets of equal value. Let us consider an optimal acquisition sequence for G. By Claim 1, we know that c will never acquire the vertices v_1, t_1, and t_2, and hence, at least one of these three vertices is in the residual set. Additionally, if c is not acquired by v_1, or if c does not have a total weight of at least $P + \frac{M}{2}$ at the time it is acquired by v_1, then the residual set will have size at least 3. Indeed, in both cases the two vertices t_1 and t_2 cannot be acquired by any neighbour, and at least one of v_2, s_1, and s_2 will remain in the residual set since none of them can be acquired by c by Claim 1. This implies that c must acquire a weight of at least $\frac{M}{2}$ from the vertices $u_1, \ldots, u_\ell$. Moreover, for similar reasons, at least one of s_1 and s_2 must be acquired by one of their neighbours, and this neighbour cannot be c, and consequently must be v_2. However, for this move to be possible, v_2 must acquire at least $\frac{M}{2}$ from its neighbours, and since v_2 cannot acquire c, then this weight of $\frac{M}{2}$ must come from the vertices $u_1, \ldots, u_\ell$. Hence, if we denote by S_1 the weights x_i which are acquired by c, and S_2 the weights acquired by v_2, then the two sets have a total sum of at least $\frac{M}{2}$, and consequently, exactly $\frac{M}{2}$ since $\sum_{1 \leq i \leq \ell} x_i = M$. Hence, (S_1, S_2) is a partition of S into two subsets of equal value.

Now, let us prove the other direction, and assume that the set S can be partitioned into two subsets S_1 and S_2 of equal value. Using this partition, consider the following acquisition sequence:

1. for all $1 \leq i \leq \ell$, if $x_i \in S_1$, then c acquires u_i, otherwise, v_2 acquires u_i. After this, c has a weight of $P + \frac{M}{2}$, and v_2 has a weight of $Q + \frac{M}{2}$.
2. v_1 acquires c, and then both t_1 and t_2;
3. v_2 acquires both s_1 and s_2.

The residual set for this acquisition sequence has size two, which proves that $a_t(G,w) \leq 2$ and proves the theorem. □

3.4 Planar graphs

In this section, we study the weighted total acquisition number in planar graphs. We show that the Binary WTA problem is NP-complete in wheels, a simple class of planar graphs. However, similarly to trees, we show that the problem can be solved in polynomial time for instances with a bounded weighted total acquisition number. Finally, we show that for a slightly larger class of graphs, namely planar graphs of pathwidth 2, the Binary 2-WTA problem is NP-complete.

Theorem 7. *For all integers $k \geq 1$, the* BINARY k-WTA *problem can be solved in $n^{O(k)}$ time on wheels.*

Proof. Let $G = (V, E)$ be a wheel with center c, w be a weight function on G, and k a fixed positive integer. The main idea of the algorithm consists in proving that there are only at most $n^{O(k)}$ possible candidates we need to consider for the acquisition forest that we are searching for.

First, notice that there are at most $n^{O(k)}$ partitions of G into at most k vertex-disjoint connected components. Indeed, a connected component of G that does not contain c is a path and can be identified by its extremities. So, there are $O(n^2)$ such components. Consider now a connected component C which contains c. Since $G[V \setminus C]$ contains at most $k-1$ connected components and those components can be identified by their extremities, C can be identified by at most $2(k-1)$ vertices. Thus, there are $n^{O(k)}$ connected components that can contain c and $n^{O(k)}$ ways to partition into k connected components.

Hence, to complete the algorithm it is sufficient to be able to check in polynomial time if each component of a given partition has a weighted total acquisition number of 1. If the component is a path, then this can be done using the algorithm from [8]. Hence, we only need to consider the component which contains the center of the wheel. We will show that, for this component, there is only a polynomial number of acquisition forests to consider. We know that if we remove the vertex c, this component is a collection of paths. Moreover, without loss of generality, we can assume that, for all but one of these paths, a solution acquisition forest does not contain edges of the path (the exception being the path that will contain the residual vertex, if it is not c). Indeed, if there is an acquisition move from u to v, and then from v to c, then the first move can be replaced by an acquisition move from u to c directly. By repeating this simplification, we can assume that c acquires all the vertices in each of the paths of the component, except possibly one. Hence, once we have chosen which vertex will be residual, and which neighbour c will transfer its tokens to, the acquisition forest satisfying this property is fixed. Hence, since there are at most $O(n^2)$ possibilities for these two choices, we can check in polynomial time if the component containing the center has a weighted total acquisition number of 1. This completes the proof of the existence of such an algorithm, and proves the theorem. □

If the size of the desired residual set is not fixed, the problem becomes NP-complete.

Theorem 8. *The* BINARY WTA *problem is NP-complete in wheels.*

Proof. The hardness of the problem is shown by a reduction from PARTITION. Given an instance $S = \{x_1, \ldots, x_\ell\}$ of the PARTITION problem, we construct, in polynomial time, a graph G with a weight function w such that $a_t(G, w) \leq \ell + 2$ if and only if S can be partitioned into two subsets S_1 and S_2 of equal value.

The graph G will be composed of a central vertex c, and $2\ell + 7$ vertices $v_1, \ldots, v_{2\ell+1}, u_1, s_1, s_2, s_3, s_4, u_2$ on the outer cycle in this order. Let $M =$

$\sum_{1\leq i\leq \ell} x_i$, and let Q, R be integers such that $Q > 50M$, and $R > 50Q$. The weight function w is built such that for all $1 \leq i \leq \ell$:

$$w(v_1) = R \qquad w(v_{2i+1}) = R \qquad w(v_{2i}) = x_i$$
$$w(c) = Q \qquad w(u_1) = w(u_2) = M \qquad w(s_1) = 2Q + 4M$$
$$w(s_2) = Q + \frac{7M}{2} \qquad w(s_3) = 4Q + 8M \qquad w(s_4) = 8Q + 15M$$

The vertices with weight R act as some sort of separator. Since their weight is too large, they will never be acquired and they remain in the residual set at the end of the acquisition sequence. The main idea is that the center c will need to acquire exactly $\frac{M}{2}$ tokens if we want only one residual vertex for the vertices s_1, s_2, s_3, s_4. See Figure 5 for an illustration of the construction.

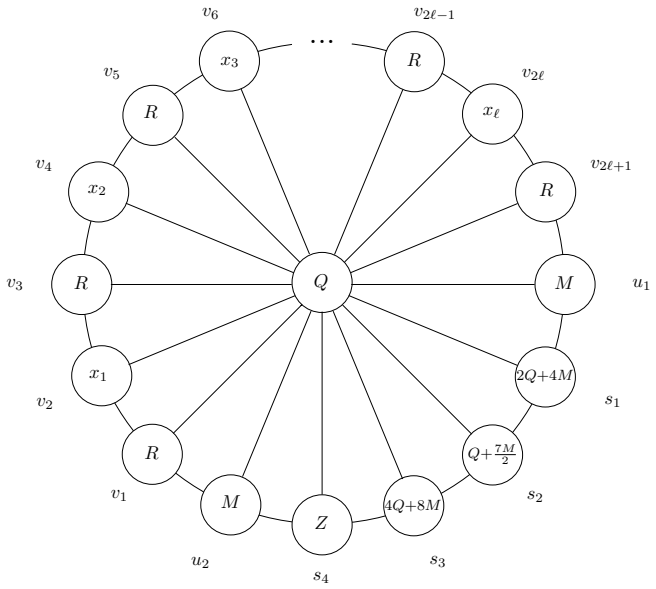

Figure 5: Construction of the wheel graph G from the proof of Theorem 8, where $Z = 8Q + 15M$.

Let us first assume that $a_t(G) \leq \ell + 2$, and let us prove that there is a partition of S into two subsets of equal value. Since R is taken sufficiently large, we know that the vertices v_1 and v_{2i+1} will never be acquired and remain in any residual set at the end of an acquisition sequence. Moreover, the center c can acquire a weight of at most $3M$ from the vertices v_{2i} and u_1, u_2, and consequently will never be able to acquire any of the s_i. Similarly, by the choice of Q, we also know that the vertices u_i will not be able to acquire any of their neighbouring s_i. Since there are at most $\ell+2$ vertices in the residual set at the end, and $\ell+1$

of these vertices are v_1 and the v_{2i+1}, this implies that the vertices $s_1, \ldots, s_4$ must be ultimately acquired by a single vertex. This vertex will have a weight of at least $\sum_{1 \leq i \leq 4} w(s_i) \geq 15Q$ at the end. Since the s_i cannot be acquired by other vertices that are not s_i, and the initial weights of s_3 and s_4 combined are much larger than those of s_2 and s_1 (even combined and after receiving all the other possible weights they could acquire), the vertex which acquires the vertices s_i can only be either s_3 or s_4.

This implies that s_1 must be acquired by s_2, which is only possible if s_2 first acquires at least $Q + \frac{M}{2}$ tokens from another vertex. This other vertex can only be the central vertex c. Hence, after acquiring both c and s_1, the vertex s_2 has a weight of at least $4Q + 8M = w(s_3)$. Moreover, since the residual vertex is either s_3 or s_4, then s_2 must be acquired by s_3, which is only possible if its weight at this point is at most $w(s_3) = 4Q + 8M$. This implies that s_2 acquired at most $Q + \frac{M}{2}$ from c, and consequently acquired exactly this quantity from c.

Hence, if we denote by S_1 the set of weights acquired by c, and $S_2 = S \setminus S_1$, then these two sets form a partition of S, and by the construction of S_1 and the argument above, we have $\sum_{x \in S_1} x = \frac{M}{2}$, which proves that the two subsets have equal value.

Now for the other direction, assume that S can be partitioned into two subsets S_1 and S_2 such that $\sum_{y \in S_1} y = \sum_{z \in S_2} z = \frac{M}{2}$, and let us construct an acquisition sequence for G. Using this partition, the central vertex c can acquire a total weight of exactly $\frac{M}{2}$ by acquiring all the x_i for $x_i \in S_1$. All the remaining vertices v_{2i} which were not acquired by c at this point, as well as u_1 and u_2 are acquired by one of their neighbours with weight R. Then, apply the following acquisition moves in this order:

1. s_2 acquires c and has a new weight of $2Q + 4M$;
2. s_2 acquires s_1 and now has a weight of $4Q + 8M$;
3. s_3 acquires s_2 and now has a weight of $8Q + 16M$;
4. finally, s_3 acquires s_4.

At the end of this acquisition sequence the only remaining vertices are s_3, v_1, and the v_{2i+1}, which proves that $a_t(G, w) \leq \ell + 2$. □

Theorem 9. *The* Binary 1-WTA *problem is NP-complete in planar graphs of pathwidth 2.*

Proof. To show that the problem is NP-hard, we reduce from the Partition problem. Given an instance $S = \{x_1, \ldots, x_\ell\}$ for Partition, we construct, in polynomial time, an instance G with a weight function w such that $a_t(G, w) \leq 1$ if and only if S can be partitioned into two subsets S_1 and S_2 of equal value.

Let $M = \sum_{1 \leq i \leq \ell} x_i$, and let Q and R be two integers such that $Q > M$ and $R > 10Q$. The graph G and the weight function w consist of:

- $\ell + 1$ vertices $v_1, \ldots, v_\ell, t$ forming an independent set, with $w(v_i) = x_i$ for all $1 \leq i \leq \ell$, and $w(t) = R$;
- two vertices u_1 and u_2 adjacent to t and all the vertices in $v_1, \ldots, v_\ell$, and such that $w(u_1) = w(u_2) = Q$;
- two vertices s_1 and s_2 adjacent to u_1 and u_2 respectively, with $w(s_1) = w(s_2) = Q + \frac{M}{2}$.

We can see in Figure 6 that G is planar, and it can be easily checked that it also has pathwidth 2.

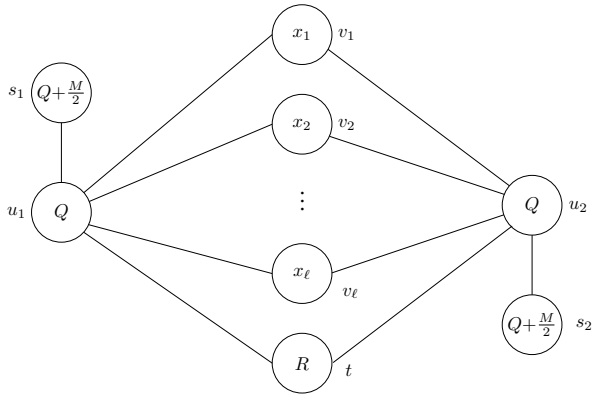

Figure 6: Construction of the graph G of pathwidth 2 from the proof of Theorem 9.

Let us first assume that $a_t(G, w) = 1$, and let us prove that S can be partitioned into two sets of equal value. Consider an optimal acquisition sequence for G. Note that since R, the weight of t, is sufficiently large, and the size of the residual set is 1, this residual set must be equal to $\{t\}$. This implies that both s_1 and s_2 must be acquired by their only neighbour u_1 and u_2 respectively. However, for this to happen, both u_1 and u_2 must first acquire a weight of at least $\frac{M}{2}$, and since $\sum_{1 \leq i \leq \ell} x_i = M$, they can only acquire exactly $\frac{M}{2}$ each. Hence, if S_i are the weights acquired by u_i from the vertices $v_1, \ldots, v_\ell$, then by the argument above, (S_1, S_2) is a partition of S where the two parts sum to the same value.

Now for the other direction, assume that S can be partitioned into two subsets S_1 and S_2 such that $\sum_{y \in S_1} y = \sum_{z \in S_2} z = \frac{M}{2}$. Consider the following acquisition sequence:

1. u_j acquires all the vertices v_i with $x_i \in S_j$, for all $1 \leq i \leq \ell$ and $1 \leq j \leq 2$;
2. at this point, u_1 and u_2 both have a weight of $Q + \frac{M}{2}$, and they can acquire s_1 and s_2 respectively;
3. finally, t acquires both u_1 and u_2.

Clearly, the residual set at the end of this acquisition sequence has size 1, which proves that $a_t(G, w) \leq 1$. □

Corollary 1. *The* Binary 1-WTA *problem is NP-complete in unit interval graphs.*

Proof. Observe that in the construction of the graph G in the proof of Theorem 9, we can replace the independent set formed by the vertices $v_1, \ldots, v_\ell, t$ by a clique, and all the arguments still work the same. By doing this, the graph G becomes a unit interval graph, which proves the theorem. □

4 Unary WTA

In the previous section, we studied the complexity of Binary WTA, and showed that this problem is hard, even for very simple classes of graphs. In this section, we are interested in the unary version of the problem. Unlike the binary version, this problem is more tractable. In particular, there are cases where a dynamic programming approach can be used to compute the weighted total acquisition number of the graph. In this section, we give two dynamic programming algorithms. The first one works for graphs of bounded treewidth, for which we show that if the maximum degree is also bounded, then there is a polynomial-time algorithm. On the other hand, if the maximum degree is not bounded, the algorithm still runs in time at most $W^{O(\log W)}$, where W is the total sum of the weights, which shows in particular that the problem is unlikely to be NP-hard in this case. This contrasts with our results from the previous section where the problem was shown to be NP-hard on trees, wheels, and planar graphs of bounded pathwidth, all of which are classes of graphs with bounded treewidth. Our second algorithm applies to trees and computes the weighted total acquisition number in time $O(f(\Delta) \cdot (poly)(W))$, where f is a computable function, and Δ is the maximum degree of the graph. This shows that the problem is FPT in trees when parameterized by the maximum degree of the tree.

On the negative side, we show that even in its unary version, the problem remains NP-complete in trivially perfect graphs. Moreover, even deciding if the weighted total acquisition number is equal to 1 is hard for split graphs.

4.1 Bounded treewidth

In this subsection, we devise an algorithm for graphs of bounded treewidth. Our result is the following:

Theorem 10. *The* Unary WTA *problem can be solved in graphs of bounded treewidth G in time at most $W^{O(\mathrm{tw}(G)\min(\Delta(G),\mathrm{tw}(G)+\log W))}$.*

Note that this result gives a polynomial-time algorithm on bounded degree trees, and a quasi-polynomial-time algorithm for general trees, which shows that Unary WTA is unlikely to be NP-hard on trees, and on graphs of bounded

treewidth in general. The question of whether the problem can be solved in polynomial time in general trees is still open.

Our algorithm will use a tree decomposition of the graph which has some additional properties. Recall that, given a graph G, a tree decomposition of G is given by a tree T, where each node of T is labelled by a subset of vertices of G, and each vertex induces a connected component of the tree. A tree decomposition is *nice* if T is rooted at a node labelled by the empty set, and every node a of the tree satisfies one of the following four cases:

- a is a *leaf node* and is labelled by the empty set;
- a is a *delete node*: it has exactly one child b, and the label of b is the label of a plus one vertex;
- a is an *introduce node*: it has exactly one child b, and the label of b is the label of a minus one vertex;
- a is a *branch node*: it has exactly two children with the same label as a.

It is known that if a graph has treewidth k, then a nice tree decomposition of width k can be computed in time $2^{O(k^3 \log k)} \cdot n$ [4].

The main idea of our algorithm is to use dynamic programming on the tree decomposition of the graph to compute an optimal solution. More precisely, let T be a nice tree decomposition of the graph G. Given a node a of T, we will denote by T_a, the subgraph of G induced by the vertices which appear in the labels of the descendants of a, and by S_a, the label of a.

The algorithm will proceed by computing, for every node a, and every possible "configuration" of the solution on S_a, the optimal cost of a solution of T_a which agrees with this configuration on S_a. These configurations describe how the solution we are searching for behaves on the vertices in S_a. If the graph also had bounded degree, it would be enough to take as a configuration all the possible values taken by a solution function on the edges incident to S_a. However, when the maximum degree is not bounded, this direct approach is too expensive, and we have to rely on other tools to consider all the possible solutions around S_a. The ingredient consists in managing to regroup a certain number of partial solutions that can be completed in the same fashion.

More precisely, if we assume that some (but not all) of the edges around a vertex v have a fixed weight, we can ask how this partial solution can be extended to the remaining edges so that the weights around v satisfy the three conditions (P1), (P2), and (P3). The next subsection develops a tool to answer this question, and subsection 4.1.2 uses this tool to design a dynamic programming algorithm to compute the weighted total acquisition number for graphs of bounded treewidth.

4.1.1 Compact representation of a partial solution

In the rest of this subsection, we will identify a multiset of weights $X = \{x_1, \ldots, x_k\}$ with a weight function on the edges around some vertex v (the order of the elements in the multiset is not important), with the convention

that positive weights correspond to transfers towards v, and negative weights are transferred from v to its neighbours. Hence, by abuse of notation, we will say that the multiset satisfies one of the properties (P1), (P2), or (P3) for weight w if the weight function f corresponding to this multiset satisfies these properties for a vertex v with w tokens on v. We will denote by $\mathfrak{X}_W^k$ the set of all multisets of at most k elements of value at most W.

Given two multisets $X = \{x_1, \dots, x_l\}$ and $Y = \{y_1, \dots, y_r\}$, let $X \uplus Y$ be the union (with multiplicities) of X and Y: $X \uplus Y = \{x_1, \dots, x_l, y_1, \dots y_r\}$. Our goal is to group together multisets which can be completed in a similar way. In other words, we want to say that X and Y are similar if for all Z, $X \uplus Z$ satisfies the three properties (P1), (P2), and (P3) if and only if $Y \uplus Z$ also does. This could be made into a formal definition, but we will not need all the formalism for the algorithm, and instead focus only on the tools needed for the algorithm. Note that among the three properties, (P1) and (P2) are relatively simple, and consequently we will focus on (P3) in the rest of this subsection. Note that since the property (P3) does not depend on the number of tokens that the vertex v gives to its neighbours, we will assume without loss of generality that the multisets that we consider contain only non-negative elements.

The main property that we will need consists in observing that some of the weights in the multiset X can be grouped together, without losing any information on how X can be completed into a multiset which satisfies the property (P3). More precisely, we will consider multisets of pairs of integers (x, r) representing a group of weight, where x is the cost to acquire this group (*i.e.*, the minimum value of an element in the group), and r is the total value that will be added to the vertex v when this group is acquired. Hence, a multiset of integers $X = \{x_1, \dots x_k\}$ can be naturally converted with this extension into $\tilde{\mathbf{X}} = \{(x_1, x_1), \dots, (x_k, x_k)\}$. In the following, to make the distinction clear in the notation between multisets of integers and multisets of pairs of integers we will typeset the latter with a bold font.

The main advantage of this new notation is that we can apply the following simplification. Assume that $\mathbf{X}$ contains two pairs (x_1, r_1) and (x_2, r_2) such that $x_1 \leq x_2 \leq x_1 + r_1$, then $\mathbf{X}$ can be replaced by $\mathbf{Y}$, the multiset obtained from $\mathbf{X}$ by replacing these two pairs by $(x_1, r_1 + r_2)$. Indeed, the condition implies that as soon as the weight of the vertex v is sufficient to acquire the pair (x_1, r_1), then the pair (x_2, r_2) can be acquired immediately afterwards since v has a weight of at least x_1 before acquiring (x_1, r_1) and a weight of at least $x_1 + r_1$ after acquiring it. This simplification will be denoted by $\mathbf{X} \to \mathbf{Y}$, and we write $\mathbf{X} \to^* \mathbf{Y}$ if $\mathbf{Y}$ is obtained from $\mathbf{X}$ by a (possibly empty) sequence of such simplifications. Moreover, $\mathbf{Y}$ will be called *terminal* if no such simplification can occur. Note that, since each time a simplification is made, the number of pairs in the multiset decreases by 1, these simplifications cannot occur indefinitely. Moreover, the following lemma shows that the order by which these simplifications are made is not important, since the terminal multiset that we obtain at the end is unique.

Lemma 3. *Given a multiset of pairs* $\mathbf{X}$*, there is a unique terminal value* $\mathbf{X}^*$ *such that* $\mathbf{X} \to^* \mathbf{X}^*$.

Proof. The proof follows immediately by applying induction and using the following claim:

Claim 2. *If we have* $\mathbf{X} \to \mathbf{Y_1}$ *and* $\mathbf{X} \to \mathbf{Y_2}$ *with* $\mathbf{Y_1} \neq \mathbf{Y_2}$*, then there exists* $\mathbf{Z}$ *such that* $\mathbf{Y_1} \to \mathbf{Z}$ *and* $\mathbf{Y_2} \to \mathbf{Z}$*.*

Proof of claim. Assume that the transformation $\mathbf{X} \to \mathbf{Y_1}$ is made by simplifying the pairs (x_1, r_1) and (x_2, r_2), and the transformation $\mathbf{X} \to \mathbf{Y_2}$ simplifies (x_3, r_3) and (x_4, r_4). If these pairs are all different, then we can simply take $\mathbf{Z}$ to be the sequence obtained from $\mathbf{Y_1}$ by simplifying the pairs (x_3, r_3) and (x_4, r_4). Hence, we can assume without loss of generality that $(x_1, r_1) = (x_4, r_4)$. Let $\mathbf{Z}$ be the multiset obtained from $\mathbf{X}$ by replacing these three pairs by $(\min(x_1, x_2, x_3), r_1 + r_2 + r_3)$. We want to show that $Y_1 \to Z$ and $Y_2 \to Z$. There are four cases:

- $x_1 \leq x_2$ and $x_1 \leq x_3$, then from $\mathbf{Y_1}$, the two pairs $(x_1, r_1 + r_2)$ and (x_3, r_3) can be replaced by $(x_1, r_1 + r_2 + r_3)$ since $x_1 \leq x_3 \leq x_1 + r_1 \leq x_1 + r_1 + r_2$ by the assumption that $\mathbf{X} \to \mathbf{Y_2}$, and consequently, $\mathbf{Y_1} \to \mathbf{Z}$. By symmetry, we also have $\mathbf{Y_2} \to \mathbf{Z}$.
- $x_2 \leq x_1$ and $x_3 \leq x_1$, then we know that $x_1 \leq x_2 + r_2$ and $x_1 \leq x_3 + r_3$, and hence, $x_3 \leq x_1 \leq x_2 + r_2 \leq x_2 + r_1 + r_2$. Additionally, we have $x_2 \leq x_1 \leq x_3 + r_3$, and since either $x_2 \leq x_3$ or $x_3 \leq x_2$, it follows that, from $\mathbf{Y_1}$, the pairs $(x_2, r_1 + r_2)$ and (x_3, r_3) can be simplified together to obtain $\mathbf{Z}$, and hence, we have $\mathbf{Y_1} \to \mathbf{Z}$. By symmetry, we also have $\mathbf{Y_2} \to \mathbf{Z}$.
- $x_2 \leq x_1$ and $x_1 \leq x_3$, then $x_2 \leq x_3$, and $x_3 \leq x_1 + r_1 \leq x_2 + r_1 + r_2$. Hence, the two pairs $(x_2, r_1 + r_2)$ and (x_3, r_3) in $\mathbf{Y_1}$ can be simplified, and $\mathbf{Y_1} \to \mathbf{Z}$. In a similar way, we have $x_2 \leq x_1$ and $x_1 \leq x_2 + r_2$, and hence, the two pairs (x_2, r_2) and $(x_1, r_1 + r_3)$ in $\mathbf{Y_2}$ can be simplified, and hence, $\mathbf{Y_2} \to \mathbf{Z}$.
- $x_1 \leq x_2$ and $x_3 \leq x_1$, then this is symmetric to the case above. ◇ □

We can thus define the function σ, which associates, to a given multiset of pairs of integers $\mathbf{Y}$, the unique terminal $\mathbf{Y}^*$ such that $\mathbf{Y} \to^* \mathbf{Y}^*$. By extension, if X is a multiset of integers, then we write $\sigma(X) = \sigma(\tilde{\mathbf{X}})$. Note that σ satisfies the following property that we will use several times during the proofs:

Observation 1. *For all multisets* X *and* Y*, the following equality holds:*

$$\sigma(X \uplus Y) = \sigma(\sigma(X) \uplus \sigma(Y)).$$

Another observation concerns the size of a terminal value, which explains why this representation can be used as a compressed way to represent a partial solution around a vertex:

Observation 2. *Let* X *be a multiset of elements with values upper bounded by* W*, then*

$$|\sigma(X)| \leq \min(|X|, \log W + 1).$$

Proof. The fact that $|\sigma(X)| \leq |X|$ follows immediately from the fact that the simplification decreases the size of the multiset. Hence, we only need to show that $|\sigma(X)| \leq \log W + 1$. Let $(x_1, r_1), \ldots, (x_k, r_k)$ be the pairs in $\sigma(X)$ ordered by increasing x_i, and with $k = |\sigma(X)|$. We can assume that $r_i \geq x_i$ for all $1 \leq i \leq k$ since this property is true for $\tilde{\mathbf{X}}$ and is preserved by the simplification observation. Moreover, since $\sigma(X)$ is terminal, then, for all $1 \leq i < k$, we also have that $x_{i+1} > x_i + r_i \geq 2x_i$, from which it follows immediately that $x_k \geq 2^{k-1}$. Since we also have $x_k \leq W$, this implies $k \leq \log W + 1$. □

We can now describe how this simplification procedure is related to the property (P3).

Lemma 4. *The multiset X satisfies property (P3) for weight w if and only if $\sigma(\tilde{\mathbf{X}} \uplus \{(0, w)\}) = \{(0, A)\}$ where $A = w + \sum_{x \in X} x$.*

Proof. The result is proved by induction on the size of X. If X is empty, then the result is always true, and there is nothing to prove. Hence, we can assume that X contains at least one element.

First assume that X satisfies property (P3) for the weight w, and let us show that $\sigma(\tilde{\mathbf{X}} \uplus \{(0, w)\}) = \{(0, A)\}$. Let $x_1, \ldots, x_k$ be the elements of X in increasing order. Since X satisfies property (P3), then in particular it follows that $x_1 \leq w$. Consequently, if $Y = X \setminus \{x_1\}$, then there is a simplification from $\tilde{\mathbf{X}} \uplus \{(0, w)\}$ to $\tilde{\mathbf{Y}} \uplus \{(0, w + x_1)\}$, and consequently $\sigma(\tilde{\mathbf{X}} \uplus \{(0, w)\}) = \sigma(\tilde{\mathbf{Y}} \uplus \{(0, w + x_1)\})$. Moreover, we know that for any $1 < i \leq k$ we have:

$$(w + x_1) + \sum_{1<j<i} x_j = w + \sum_{j<i} x_j \geq x_i \, .$$

Consequently, Y satisfies the property (P3) for the weight $w + x_1$, hence using the induction hypothesis, it follows that $\sigma(\tilde{\mathbf{Y}} \uplus \{(0, w + x_1)\}) = \{(0, A)\}$ where $A = w + x_1 + \sum_{y \in Y} y = w + \sum_{x \in X} x$.

Let us now prove the other direction and assume that $\sigma(\tilde{\mathbf{X}} \uplus \{(0, w)\}) = \{(0, A)\}$, and let us show that X satisfies the property (P3). Let $x_1 = \min(X)$, then we must have $x_1 \leq w$. Indeed, if this was not the case, then the pair $(0, w)$ could never be simplified with any of the other pairs, which would contradict the assumption that $\sigma(\tilde{\mathbf{X}} + \{(0, w)\})$ contains a single pair. Hence, it follows that if $Y = X \setminus \{x_1\}$, then there is a simplification from $\tilde{\mathbf{X}} \uplus \{(0, w)\}$ to $\tilde{\mathbf{Y}} \uplus \{(0, w + x_1)\}$. By consequence, it follows that $\sigma(\tilde{\mathbf{Y}} \uplus \{(0, w + x_1)\}) = \{(0, A)\}$, and using the induction hypothesis, this implies that Y satisfies the property (P3) for the weight $x_1 + w$. This means that for all $1 \leq i \leq k$, we have:

$$w + \sum_{j<i} x_j = w + x_1 + \sum_{1<j<i} x_j \geq x_i \, .$$

And since we also have $x_1 \leq w$, then X satisfies (P3) for the weight w. □

We now have all the tools that we need to describe the algorithm.

4.1.2 Algorithm

The algorithm proceeds using dynamic programming on the tree-decomposition of the graph G. Let a be a node in the tree decomposition of G, and S_a, its associated bag. A *configuration* for the node a is a triple (g, α, μ) where:

- g is a weight function on the edges of $G[S_a]$, which represents the restriction of the solution to these edges;
- α is a function which associates, to each vertex $v \in S_a$, an integer with $0 \leq \alpha(v) \leq W$ representing the number of tokens v gives to one of its neighbours in $T_a \setminus S_a$ (if any);
- μ is a function which associates, to each vertex v in S_a, a terminal multiset with a size of at most $\min(\Delta(G), \log(W) + 1)$. The value $\mu(v)$ represents how a solution behaves on the edges of the form $v_i u$ with $u \in T_a \setminus S_a$.

Note that the upper bound on the length of the values $\mu(v)$ in the configuration follows from the fact that larger multisets cannot occur due to the upper bound on the length of values of the form $\sigma(X)$ given by Observation 2. We will denote by $\mathcal{C}_a$, the set of configurations for the node a. Note that, by definition, we immediately have the following:

Observation 3. *The number of configurations of a node a is at most $|C_a| \leq W^{O(\mathrm{tw}(G)\min(\Delta(G),\mathrm{tw}(G)+\log W))}$ for any node a.*

Proof. Note that since there are at most $\mathrm{tw}(G)\min(\mathrm{tw}(G), \Delta(G))$ edges in $G[S_a]$, then there are at most $(2W+1)^{\mathrm{tw}(G)\min(\mathrm{tw}(G),\Delta(G))}$ possible values for g. Similarly, there are at most $(W+1)^{\mathrm{tw}(G)+1}$ possible values for α. Lastly, for each vertex v in S_a, $\mu(v)$ is a multiset of size at most $\min(\Delta(G), \log W + 1)$, and each element in the multiset is a pair of integers between 0 and W. Therefore, there are at most $(W+1)^{2(\mathrm{tw}(G)+1)\min(\Delta(G),\log W+1)}$ possible values for μ. All in all, this gives an upper bound of $W^{O(\mathrm{tw}(G)\min(\mathrm{tw}(G),\Delta(G))+\mathrm{tw}(G)\min(\Delta(G),\log W))} = W^{O(\mathrm{tw}(G)\min(\Delta(G),\mathrm{tw}(G)+\log W))}$ for the number of possible configurations. □

A weight function f on the edges of T_a is a *partial solution* for T_a if it satisfies the property (P1) for all the vertices in T_a, and properties (P2) and (P3) for all vertices in $T_a \setminus S_a$. We denote by $\mathcal{F}_a$ the set of partial solutions for T_a. A partial solution for T_a is *compatible* with a given configuration (g, α, μ) of a if:

(C1) $f|_{S_a} = g$;

(C2) for all $v \in S_a$, we have $\alpha(v) = \sum_{u \in T_a \setminus S_a} f^+(vu)$ (note that at most one term in this sum can be positive).

(C3) for all $v \in S_a$ we have $\mu(v) = \sigma(\{f(uv_i),\ u \in N_G(v) \cap T_a \setminus S_a\})$.

Note that these configurations define a partition of the set of partial solutions $\mathcal{F}_a$. Indeed, it is clear that a partial solution f cannot be compatible with more than one configuration, and also, f is always compatible with the configuration $c_f^a = (g_f, \alpha_f, \mu_f)$, where g_f, α_f, and μ_f are taken such that the three conditions above are satisfied.

The cost of such a weight function f will be the number of sinks for f in $T_a \setminus S_a$. The algorithm will proceed using dynamic programming by computing, for every node a of the tree decomposition and every configuration c of the node, the minimal cost of a weight function f compatible with this configuration, with the convention that this cost is $+\infty$ if no such weight function exists. The result computed by the algorithm will be denoted by $\text{cost}(a, c)$. The algorithm is the following:

Input:
- a: a node of the tree decomposition
- $c = (g, \alpha, \mu)$ a configuration of a.

Output: $\text{cost}(a, c)$, the minimum cost of a solution compatible with c.

---

1. a **is a leaf node**: return 0.
2. a **is a branch node** with children b_1 and b_2.
   $result \leftarrow +\infty$
   Iterate over all $(c_1, c_2) \in \mathcal{C}_{b_1} \times \mathcal{C}_{b_2}$.
   - 2.1. $(g_i, \alpha_i, \mu_i) \leftarrow c_i$ for $i \in \{1, 2\}$
   - 2.2. Discard (c_1, c_2) and continue if one of the following does not hold:
     - i. $g_1 = g_2$
     - ii. $\forall v \in S_a,\ \alpha(v) = \alpha_1(v) + \alpha_2(v)$
     - iii. $\forall v \in S_a,\ \mu(v) = \sigma(\mu_1(v) \uplus \mu_2(v))$
     - iv. $\forall v \in S_a,\ \alpha_1(v)\alpha_2(v) = 0$
   - 2.3. $result \leftarrow \min(result, \text{cost}(b_1, c_1) + \text{cost}(b_2, c_2))$

   return $result$.
3. a **is a delete node** with child b, and v is the deleted vertex.
   $result \leftarrow +\infty$
   Iterate over all $c' = (g', \alpha', \mu') \in \mathcal{C}_b$
   - 3.1. $X_v \leftarrow \{g'(uv),\ u \in S_a \setminus v\}$
   - 3.2. $\beta_v \leftarrow \alpha'(v) + \sum_{u \in S_a} g'^{+}(vu)$
   - 3.3. Discard c' and continue if one of the following does not hold:
     - i. $g'|_{G[S_a]} = g$
     - ii. $\forall u \in S_a,\ \alpha(u) = \alpha'(u) + g'^{+}(uv)$
     - iii. $\forall u \in S_a,\ \mu(u) = \sigma(\mu'(u) + \{(y, y)\})$ with $y = g'^{+}(vu)$
     - iv. $\sigma(\mu(v) \uplus \tilde{\mathbf{X}}_{\mathbf{v}}) = \{(0, r)\}$ for some r
     - v. $\beta_v \in \{0, r\}$
   - 3.4. $result \leftarrow \min(result, \text{cost}(b, c') + \mathbf{1}_{\beta_v = 0})$

   return $result$
4. a **is an introduce node** with child b, and v is the introduced vertex.
   - 4.1. If $\alpha(v) > 0$ or $\mu(v) \neq \emptyset$, then return $+\infty$
   - 4.2. return $\text{cost}(b, (g|_{S_b}, \alpha|_{S_b}, \mu|_{S_b}))$

The algorithm then returns $\text{cost}(r, \emptyset)$ where r is the root of the tree decomposition.

Proof of Theorem 10. To prove the theorem, it only remains to show that the algorithm described above is correct and has the desired runtime.

Correctness. To prove that the algorithm is correct, we will first show that for all nodes a, and all configurations c of a, the algorithm computes the minimal cost of a solution compatible with c. The proof for this result proceeds by induction on the size of the subtree of T rooted in a. If a has no children, then a is a leaf and S_a is empty. In this case, there is only one configuration for a, that is the empty configuration and the algorithm gives a cost of 0 for this configuration. Let us assume that the property is true for all subtrees of size at most K, and let a be a node such that the subtree rooted at a has size $K+1$. We distinguish three cases depending on the type of the node a.

a is a branch node. Let $c = (g, \alpha, \mu)$ be a configuration of a. First, assume that there is a solution f for T_a compatible with c and with a cost of A. We will show that the result returned by the algorithm is at most A. Let b_1 and b_2 be the two children of a, and let f_1 and f_2 be the restriction of f to $G[T_{b_1}]$ and $G[T_{b_2}]$ respectively. By definition, for $i \in \{1, 2\}$ we know that f_i is compatible with the configuration $c_{f_i} = (g_i, \alpha_i, \mu_i)$, and additionally, the cost of f is exactly $A_1 + A_2$ where A_i is the cost of f_i since the two sets $T_{b_i} \setminus S_{b_i}$ for $i \in \{1, 2\}$ form a partition of $T_a \setminus S_a$. Hence, we only need to prove that the pair (c_{f_1}, c_{f_2}) is not discarded by the algorithm at step 2.2.. In other words, we need to show that the pair satisfies all four properties i. through iv. which are checked at this step:

i. We clearly have $g_1 = g_2$ since f_1 and f_2 are both restrictions of the same function f.

ii. Since f is compatible with c, we have the following for any $v \in S_a$:

$$\begin{aligned}\alpha(v) &= \sum_{u \in T_a \setminus S_a} f^+(vu) \\ &= \sum_{u \in T_{b_1} \setminus S_a} f^+(vu) + \sum_{u \in T_{b_2} \setminus S_a} f^+(vu) \\ &= \alpha_1(v) + \alpha_2(v)\ .\end{aligned}$$

iii. In a similar way, we have:

$$\begin{aligned}\mu(v) &= \sigma(\{f(uv),\ u \in T_a \setminus S_a\}) \\ &= \sigma(\{f(uv),\ u \in T_{b_1} \setminus S_a\} + \{f(uv),\ u \in T_{b_2} \setminus S_a\}) \\ &= \sigma(\sigma(\{f(uv),\ u \in T_{b_1} \setminus S_a\}) + \sigma(\{f(uv),\ u \in T_{b_2} \setminus S_a\})) \\ &= \sigma(\mu_1(v) + \mu_2(v))\ .\end{aligned}$$

iv. Finally, since f is compatible with c, the property (P1) is satisfied for all the vertices in $v \in S_a$, in particular, there is at most one vertex $u \in T_a \setminus S_a$ with $f(vu) > 0$. Hence, one of $\alpha_1(v)$ or $\alpha_2(v)$ is equal to zero.

Since the four properties are satisfied for the pair (c_1, c_2), this pair is not discarded. Using the induction hypothesis, the result returned by the algorithm on the input (b_i, c_i) is at most A_i, and the result returned by the algorithm on the input (a, c) is at most $A_1 + A_2 = A$.

Let us now prove the converse that if the algorithm returns A, then there is a function f compatible with c with a cost of A. Let (c_1, c_2) be the pair which achieves the minimum in the iteration of step 2.. Note that since the pair (c_1, c_2) was not discarded at step 2.2., the properties i. through iv. are satisfied. Let A_1 and A_2 be the result of the algorithm on these two configurations. Using the induction hypothesis, there exist functions f_1 and f_2 compatible with c_1 and c_2 respectively, which achieve a cost of A_1 and A_2 respectively. Let f be the function such that $f(e) = f_1(e)$ if $e \in G[T_{b_1}]$ and $f(e) = f_2(e)$ if $e \in G[T_{b_2}]$. Note that this function is well-defined since f_1 and f_2 agree on the edges of $G[S_a]$ since the condition i. is satisfied. The cost of f is equal to the cost of f_1 plus the cost of f_2, hence, to prove the result, we only need to show that f is a valid solution and is compatible with c. Let us first show that f is a valid solution. Clearly, the three properties (P1), (P2), and (P3) are satisfied for all the vertices of $T_a \setminus S_a$ since f_1 and f_2 are valid solutions. Moreover, property (P1) is also satisfied for all the vertices in S_a by the condition iv. Furthermore, we can check that f satisfies the three properties to make it compatible with c:

- f agrees with g on S_a by construction,
- moreover, by the condition ii. we have for all $v \in S_a$:

$$\begin{aligned}\alpha(v) &= \alpha_1(v) + \alpha_2(v) \\ &= \sum_{u \in T_{b_1} \setminus S_a} f_1^+(vu) + \sum_{u \in T_{b_2} \setminus S_a} f_2^+(vu) \\ &= \sum_{u \in T_a \setminus S_a} f^+(vu)\end{aligned}$$

- and finally, using condition iii., we have for all $v \in S_a$:

$$\begin{aligned}\mu(v) &= \sigma(\mu_1(v) + \mu_2(v)) \\ &= \sigma(\sigma(\{f_1(uv),\ u \in T_{b_1} \setminus S_{b_1}\}) \uplus \sigma(\{f_2(uv),\ u \in T_{b_2} \setminus S_{b_2}\})) \\ &= \sigma(\{f(uv),\ u \in T_a \setminus S_a\})\end{aligned}$$

This concludes the induction step in the case where a is a branch node.

a is a delete node. Let b the child of a, and v the deleted vertex, *i.e.*, $\{v\} = S_b \setminus S_a$. Let $c = (g, \alpha, \mu)$ be a configuration of a, and let us first assume that there is a solution f compatible with c for T_a compatible with c with a cost of A. The cost of f for the node a is equal to the cost of f for the node b plus one if v is a sink in f. Let $(g', \alpha', \mu') = c_f^b$ be the configuration of f for the node b. To show that the algorithm returns a result of at most A, we only need to prove that the configuration c_f^b is not discarded at step 3.3.. We check the five properties one by one:

i. is clearly true by construction.

ii. By definition, for any $u \in S_a$, we have:

$$\begin{aligned}\alpha(u) &= \sum_{x \in T_a \setminus S_a} f^+(ux) \\ &= f^+(uv) + \sum_{x \in T_b \setminus S_b} f^+(ux) \\ &= f^+(uv) + \alpha'(u)\end{aligned}$$

since $T_a \setminus S_a = T_b \setminus S_b \cup \{v\}$.

iii. Similarly, we have for all $u \in S_a$:

$$\begin{aligned}\mu(u) &= \sigma(\{f(xu), x \in T_a \setminus S_a\}) \\ &= \sigma(\{f(xu), x \in T_b \setminus S_b\} \uplus \{f(vu)\}) \\ &= \sigma(\sigma(\{f(xu), x \in T_b \setminus S_b\}) \uplus \sigma(\{f(vu)\})) \\ &= \sigma(\mu'(u) \uplus \{(y, y)\})\end{aligned}$$

where $y = f^+(vu)$.

iv. Since f is compatible with c, in particular, it satisfies the property (P1) for any vertex in $S_a \setminus T_a$, and in particular for the vertex v. By Lemma 4, this implies that $\sigma(\sigma(\{f(uv), u \in T_a\}) \uplus \{(0, w)\})$ contains a single pair of the form $(0, r)$, where $r = w + \sum_{u \in T_a} f^+(uv)$.

v. Since f' is compatible with c', it follows that:

$$\begin{aligned}\beta_v &= \alpha'(v) + \sum_{u \in S_a} g'^+(vu) \\ &= \sum_{u \in T_a} f^+(vu)\end{aligned}$$

Since f is compatible with c, in particular it satisfies the condition (P2) at the vertex v, which implies that either $\beta_v = 0$, or $\beta_v = r$.

Hence, the configuration c' is not discarded by the algorithm, and it returns a result of at most A. Let us assume now for the other direction that the algorithm returns A, and let us show that there exists a solution f with the given cost. Let c' be the configuration of b which achieves the minimum in the iteration of step 3. of the algorithm. By definition, the algorithm returns $A' = A - \mathbf{1}_{\beta_v \neq 0}$ on the input (b, c'). Using the induction hypothesis, we know that there is a function f compatible with c' which achieves this cost. The cost of f for the node a is exactly A, hence, to prove the result, we only need to show that f is compatible with c. Since f is a valid solution for the node b, we know that it satisfies the three properties (P1), (P2), and (P3) for all the vertices in $T_b \setminus S_b$, and it satisfies (P1) for all the vertices in S_b. Since $T_a \setminus S_a = T_b \setminus S_b \cup \{v\}$, we only need to show that f satisfies the two properties (P2) and (P3) for the

vertex v. The property (P2) follows from the fact that β_v counts how many tokens v gives to one of its neighbours, and the fact that condition v. holds. The property (P3) is a consequence of conditions iv. and v. together with Lemma 4. This shows that f is a valid solution for the node a.

To show that f is compatible with c, we need to check that the three conditions (C1), (C2), and (C3) are satisfied.

- (C1) follows immediately from condition i. and the fact that f is compatible with c'.
- (C2) follows from condition ii., indeed this implies that for all $u \in S_a$:

$$\begin{aligned}\alpha(u) &= g'^{+}(uv) + \alpha'(u) \\ &= f^{+}(uv) + \sum_{w \in T_b \setminus S_b} f^{+}(uw) \\ &= \sum_{w \in T_a \setminus S_a} f^{+}(uw)\end{aligned}$$

- finally, (C3) is a consequence of iii. since it implies that for all $u \in S_a$:

$$\begin{aligned}\mu(u) &= \sigma(\mu'(u) \uplus \{(g'^{+}(vu), g'^{+}(vu))\}) \\ &= \sigma(\sigma(\{f^{+}(wu),\ w \in T_b \setminus S_b\}) \uplus \sigma(\{(f^{+}(vu)\})) \\ &= \sigma(\{f^{+}(wu),\ w \in T_a \setminus S_a\})\end{aligned}$$

  where the last step is obtained using Observation 1.

a is an introduce node. Let b be the child of a, and v be the introduced vertex. Let $c = (g, \alpha, \mu)$ be a configuration of a. Assume first that there exists a solution f compatible with c which achieves a cost of A, and let us show that the algorithm returns a result of at most A. Let f' be the restriction of f to $T_b = T_a \setminus v$. By definition, the cost of f for the node a is equal to the cost of f' for the node b. Additionally, the function f' is compatible with the configuration $c' = (g', \alpha', \mu')$, where g', α', μ' are the restrictions of g, α, μ to S_b. Finally, we must have $\alpha(v) = 0$ and $\mu(v) = \emptyset$ since, otherwise, there would be no function compatible with c as the vertex v has no neighbour in T_a outside of S_a. By induction, the algorithm returns a result of at most A on the function f' for the node b, and consequently, it also returns A on f for the node a.

Conversely, assume that the algorithm returns a result of A on the node a, and let us show that there exists a solution compatible with c with this cost. Let c' be the configuration obtained from c by a restriction to S_b. Since the algorithm returns $A < +\infty$, then we must have $\alpha(v) = 0$, $\mu(v) = \emptyset$, and the algorithm returns A on the input (b, c'). Using the induction hypothesis, this implies that there is a function f', compatible with c' with a cost of A. Let f be the function such that $f(uv) = g(uv)$ for all the vertices $u \in S_a$, and $f(e) = f'(e)$ for all the other edges. The function f is a valid solution for the node a since f' is a valid solution for b. Moreover, f is compatible with c since:

- f agrees with g on the vertices of S_a since f' agrees with g on $S_a \setminus v$, and by construction for the other edges.
- For all vertices $u \in S_a \setminus \{v\}$, the condition (C2) is clearly satisfied for f since it is satisfied for f', and $\alpha(v) = 0 = \sum_{u \in T_a \setminus S_a} f^+(vu)$ since v has no neighbours in $T_a \setminus S_a$.
- For all vertices $u \neq v$, the condition (C3) is clearly satisfied for f since it is satisfied for f', and $\mu(v) = \emptyset = \sigma(\{f(uv),\ u \in S_a\})$ since, again, v has no neighbours in $T_a \setminus S_a$.

This concludes the induction step in the case where a is an introduce node, and terminates the proof of the correctness of the algorithm.

Complexity. The complexity follows from the upper bound on the number of configurations in Observation 3. Indeed, for each configuration of a node, one step of the algorithm is performed. The most costly part of the algorithm is the iteration in step 2. which iterates over all pairs of configurations. In the end, the complexity of the algorithm is $O(K^3 n)$, where K is an upper bound on the number of configurations for any given node in the tree decomposition. Observation 3 implies that $K = W^{O(\mathrm{tw}(G)\min(\Delta(G),\mathrm{tw}(G)+\log W))}$, giving the required upper bound on the runtime of the algorithm. □

4.2 Trees

If we restrict our attention to trees, then the algorithm from the previous section can be slightly improved. More precisely, the algorithm above has a complexity of $W^{O(\Delta)}$ on trees of maximum degree Δ. We show that there is in fact an algorithm with a complexity of $O((\Delta+2)!W^4)$, which is a slight improvement on the previous result. In particular, it implies that Unary WTA is FPT on trees when parameterized by the maximum degree of the tree.

The algorithm again uses a dynamic programming approach to solve the problem. The main idea is to observe that if we know the order with which a given vertex acquires its children, then solving the problem just consists in finding weights for each children that sum to the desired value. This can be done in polynomial time using a dynamic programming algorithm somewhat similar to what is used for subset sum. Precisely, we consider the following problem:

Ordered Acquirable Sequence Extraction (OASE)

Inputs:
- Sets S_i for $i \leq k$ of pairs (x, c) of values and their associated costs, represented by a matrix M, where $M_{i,j} = c$ if $(j, c) \in S_i$, and $M_{i,j} = +\infty$ otherwise;
- An initial value w_0 and a target value W.

Output: a function f such that:
- $w_0 + \sum_{i<k} f(i) = W$,
- f can be acquired with this ordering: for all $i \leq k$ we have $f(i) \leq w_0 + \sum_{j<i} f(j)$.

And among all functions which satisfy these conditions, we want to find f with a minimal total cost where the total cost is defined as $\sum_{i \leq k} M_{i,f(i)}$.

Lemma 5. *The OASE problem can be solved in time $O(kW^2)$.*

Proof. The algorithm proceeds by dynamic programming by recording for each i, and for each possible value w the minimum cost of a function f on the first i sets. In other words, at step i of the dynamic programming algorithm, for each possible cost w, we have a function f such that:

- $w_0 + \sum_{j<i} f(j) = w$;
- for each $j < i$, $f(j) \leq w_0 + \sum_{\ell<j} f(\ell)$.

Among all the functions which satisfy these conditions, we record the ones for which $\sum_{j<i} M_{j,f(j)}$ is minimal. Clearly, by definition, at step k of the algorithm, the cost corresponding to the value $w = W$ is the solution of the problem. Hence, we only need to describe how to go from step i to step $i + 1$. This is done in a natural way, by simply trying all possibilities for the value $f(i + 1)$. More precisely, the cost for each w of step $i + 1$ is initialized with $+\infty$. Then, for each possible w and each possible x, the minimum cost of $w + x$ is updated to the minimum between its current value and the sum of the optimal cost for w at step i plus $M_{i,x}$.

Complexity. At each step, the algorithm iterates over all possible total values w for the previous step, and all possible values x for $f(i)$, which gives at most $O(W^2)$ possible choices to consider. Since there are k steps in total, the desired bound on the complexity of the algorithm follows. □

Note that the crucial property which makes the algorithm described above work is that we are already given the order according to which the values are acquired. Without this condition, the complexity of the problem is not known, and finding a polynomial-time algorithm for this problem could be transformed into a polynomial-time algorithm for trees. Using this result, we can show the following:

Theorem 11. *The* UNARY WTA *problem can be solved in time* $O((\Delta+2)!W^4)$ *on trees of maximum degree* Δ.

Proof. The algorithm proceeds by computing, for every vertex v of the tree, the cost of an optimal solution on the subtree rooted at this vertex, assuming that a certain amount of weight w is moved between v and its parent. The result returned by the algorithm will be simply the associated cost of the root assuming it receives nothing from its (non-existent) parent. If v is a leaf of the tree, then this cost is 1 if v receives at most $w(v)$ from its parent, 0 if v gives exactly $w(v)$ to its parent, and $+\infty$ in all other cases.

Assuming that these costs have been computed for all the children of v, the cost for v, assuming it gives or receives w from its parent is computed by first making a few guesses, and then invoking the algorithm for the OASE problem:

1. first we guess which neighbour v will transfer its weight to (assuming v does not already give the tokens to its parent), and possibly v itself if v is a sink in the solution we are looking for. We also guess how many tokens w are given to this vertex.
2. Then, for the remaining neighbours, we guess the order by which they will be acquired. Again, this is done by trying all the possibilities for the order according to which the neighbours are acquired.
3. Finally, once these choices are made, finding a solution for the tree rooted at the vertex v, knowing potential solutions for the subtrees rooted at each neighbour of v corresponds to solving an instance of OASE. The weight w possibly received from the parent can be treated as a set where the cost of a value is 0 if it is w, and $+\infty$ otherwise. The solution to the instance of OASE gives the optimal cost of a solution which satisfies the choices that were made in the previous steps.
4. Finally, among all the functions that were tried for all the possible choices, we only keep the one which has an optimal cost. Since we tried all possibilities, this is the optimal cost of a partial solution on the subgraph rooted at v where a weight of w is transferred between v and its parent.

The correctness of the algorithm follows immediately from the fact that we tried all possibilities, and the fact that the problem OASE looks for solutions which are acquirable. The only remaining point is to consider the complexity of the algorithm we just described.

Complexity. For each vertex of the tree, the algorithm first tries all possible choices of neighbours of v to which v can transfer its tokens, and, for each of these choices, the algorithm guesses how many tokens are transferred. There are at most ΔW possible choices for this step. Then, the algorithm chooses an ordering of the remaining children, for which there are $\Delta!$ possibilities. Finally, the resulting OASE instance is solved in time $O(\Delta W^2)$ using the algorithm from Lemma 5. This gives an overall complexity of $O(\Delta!\Delta^2W^3) = O((\Delta+2)!W^3)$ for each vertex of the tree, and since there are $n \leq W$ vertices in the tree, this gives the complexity claimed in the statement of the theorem. □

4.3 Split and trivially perfect graphs

In this subsection, we show that the WTA problem is strongly NP-complete, that is, even in its unary version, the problem remains NP-complete in split graphs and trivially perfect graphs.

Theorem 12. *The* Unary 1-WTA *problem is NP-complete in split graphs.*

Proof. The hardness is shown by a reduction from the 3-Partition problem. Given an instance $S = \{x_1, \ldots, x_{3\ell}\}$ for 3-Partition, we construct, in polynomial time, a split graph G with a weight function w (with weights polynomial in ℓ) such that $a_t(G, w) \leq 1$ if and only if the instance of 3-Partition has a solution.

Let $B = \frac{1}{\ell} \sum_{1 \leq i \leq 3\ell} x_i$, and take two integers Q and R such that $Q > \ell B$, and $R > 3\ell Q$. Moreover, we know that we can assume without loss of generality that $\frac{B}{4} < x_i < \frac{B}{2}$ for all $1 \leq i \leq 3\ell$. The split graph G is constructed in the following way:

- add ℓ vertices $u_1, \ldots u_\ell$ to the clique of G, with $w(u_i) = Q$;
- add 3ℓ vertices $v_1, \ldots, v_{3\ell}$ to the clique of G, with $w(v_i) = x_i$;
- add a single vertex c with weight R to the clique;
- finally, add ℓ vertices $y_1, \ldots, y_\ell$ to the independent set of G with $w(y_i) = Q + B$, and such that y_i is only adjacent to u_i.

An illustration of the construction can be found on Figure 7.

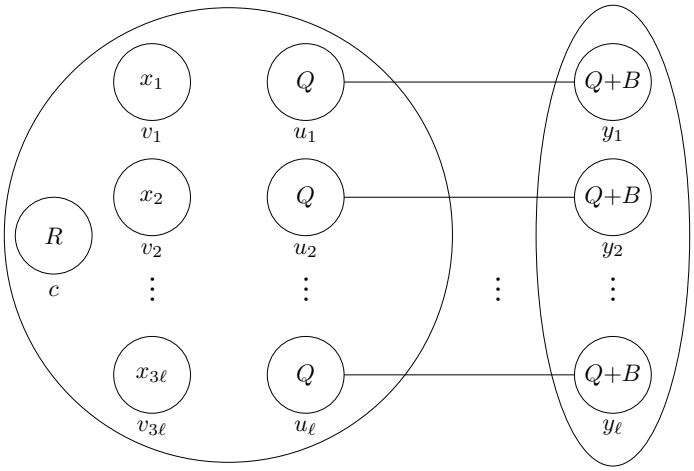

Figure 7: Construction of the split graph G from the proof of Theorem 12.

Let us first assume that $a_t(G, w) = 1$, and let us prove that the instance S of 3-Partition admits a solution. Because R is sufficiently large, we know that c will be the only vertex remaining at the end of the acquisition sequence.

This implies that, for all $1 \leq j \leq \ell$, the vertex y_j must be acquired by its only neighbour u_j. This is only possible if u_j first acquires a weight of at least B first. Moreover, this weight of B can only be acquired from the vertices v_i. Indeed, assume by contradiction that this is not the case, and that there is a vertex u_j which did not acquire B from the vertices v_i, then this vertex must acquire its weight from one other $u_{j'}$, and assume that $u_{j'}$ is the first such vertex acquired by u_j. However, by the argument above, $u_{j'}$ acquires $y_{j'}$ before being itself acquired by a neighbour, which means that u_j must reach a weight of $2Q + 2B$, which is not possible. Hence, every vertex u_j must acquire at least B from the vertices v_i, and consequently they all acquire exactly B. Hence, if we denote by S_j the set of weights acquired by u_j, then $(S_j)_{1 \leq j \leq \ell}$ is a solution to the 3-Partition instance, since $|S_j| = 3$ for all $1 \leq j \leq \ell$ due to the inequalities on the weights x_i.

Let us prove the other direction, and assume that the set S can be partitioned into ℓ subsets $S_1, \ldots, S_\ell$, such that for all $1 \leq j \leq \ell$, $\sum_{a \in S_j} a = B$. Using this partition, the acquisition sequence is the following:

1. for all $1 \leq j \leq \ell$, u_j acquires the vertices v_i whose weight is in the set S_i, which brings the weight of u_j to exactly $Q + B$;
2. then, for all $1 \leq j \leq \ell$, u_j can acquire y_j;
3. Finally, all the u_js can be acquired by c.

At the end of this sequence, $\{c\}$ is the residual set, and so, $a_t(G, w) = 1$. □

Furthermore, the Unary WTA problem is NP-complete on trivially perfect graphs when the weighted total acquisition number is not a fixed constant.

Theorem 13. *The* Unary WTA *problem is NP-complete in trivially perfect graphs.*

Proof. The hardness of the problem is proven by a reduction from 3-partition. Given an instance $S = \{x_1, \ldots, x_{3\ell}\}$ for 3-partition, we construct, in polynomial time, a graph G with a weight function w (with weights polynomial in ℓ) such that $a_t(G, w) \leq \ell$ if and only if S can be partitioned into ℓ subsets $S_1, \ldots, S_\ell$ of equal value $B = \frac{1}{\ell} \sum_{1 \leq i \leq 3\ell} x_i$. Moreover, we know that we can assume without loss of generality that $\frac{B}{4} < x_i < \frac{B}{2}$ for all $1 \leq i \leq 3\ell$.

Given an integer $Q > \ell B$, the graph G consists of:

- 3ℓ universal vertices $v_1, \ldots, v_{3\ell}$, with $w(v_i) = x_i$;
- ℓ vertex disjoint paths P_j on three vertices $u_{j,1}, u_{j,2}, u_{j,3}$, with $w(u_{j,2}) = Q$ and $w(u_{j,1}) = w(u_{j,3}) = Q + B$.

It is not difficult to see that this graph is trivially perfect as is shown in Figure 8.

First, let us assume that $a_t(G, w) \leq \ell$, and let us prove that there is a solution to the 3-partition problem. We can start by observing that since $Q > \ell B$, the vertices $u_{j,r}$ will never be acquired by any of the vertices v_i. Hence, for each $1 \leq j \leq \ell$, the tokens which are on the vertices of the path P_j never leave this

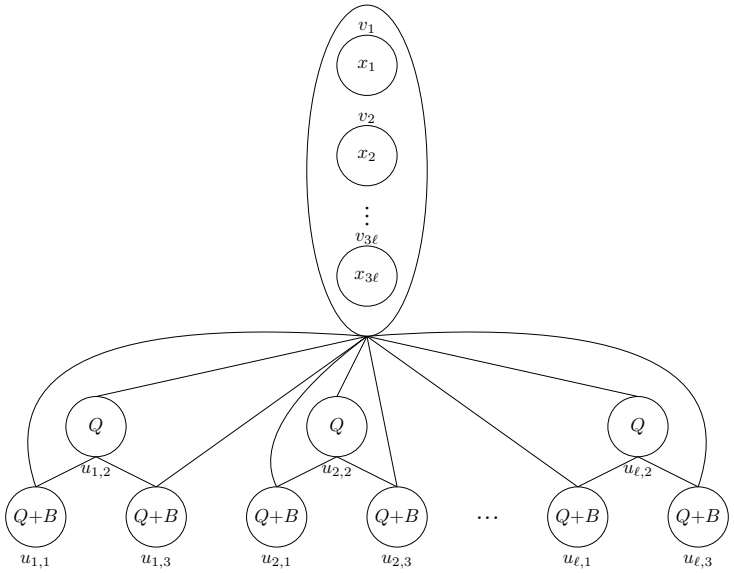

Figure 8: Construction of the trivially perfect graph G from the proof of Theorem 13.

path. Moreover, since the weighted total acquisition number of the graph G is at most ℓ, this implies that the residual set has size at most one on each path P_j.

Note that if $u_{j,2}$ does not acquire any of its two neighbours in the path P_j, then by the observations above, there must be at least two residual vertices at the end of the acquisition sequence. Hence, $u_{j,2}$ must acquire at least one of its two neighbours for all $1 \leq j \leq \ell$, which is only possible if it first acquires a weight of at least B. Hence, if we consider an acquisition sequence of cost at most ℓ, then $u_{j,2}$ acquires at least B from the vertices v_i. Moreover, since $\sum_{1 \leq i \leq 3\ell} w(v_i) = \ell \cdot B$, then the vertices $u_{j,2}$ must acquire a value of exactly B. If we denote by S_j the set of weights acquired by $u_{j,2}$ from the vertices v_i, then we have $\sum_{x_i \in S_j} x_i = B$, and $|S_j| = 3$ for all $1 \leq j \leq \ell$ due to the inequalities on the weights x_i. Hence, $(S_j)_{1 \leq j \leq \ell}$ is a solution to the 3-partition problem.

Let us now prove the other direction, and assume that $(S_j)_{1 \leq j \leq \ell}$ is a solution to the 3-partition instance S. We want to show that there is an acquisition sequence which leaves at most ℓ vertices at the end. The acquisition sequence is defined in the following way:

- for each $1 \leq j \leq \ell$, $u_{j,2}$ acquires the vertices v_i with a weight in S_j in any order;
- then, since the weight of $u_{j,2}$ is now $Q+B$, it can acquire its two neighbours $u_{j,1}$ and $u_{j,3}$.

At the end of this acquisition sequence, there is at most one vertex per path P_j with remaining tokens, which proves that $a_t(G, w) \leq \ell$. □

5 Further Work

In this article we investigated the complexity of the weighted total acquisition problem for different classes of graphs. We have seen that, in its binary version, the problem is hard even for very simple classes of graphs. One further direction of research could be to look in more detail at the parameterized complexity of the problem. For example, on trees and on wheels, we know that Binary k-WTA is in XP for the parameter k, but it is unknown if the problem is FPT. It can also be interesting to investigate the problem on other classes of graphs such as outerplanar graphs for which we conjecture that Binary k-WTA can be solved in polynomial time for constant k.

We also proved that Unary WTA is more tractable than the binary version. One question which remains open is whether the problem can be solved in polynomial time on trees or at least to look for ways to improve the $W^{O(\log(W))}$ upper bound on the time complexity of the problem for trees. Finally, it could be interesting to investigate if Unary WTA can be solved in polynomial time on other classes of graphs such as complete bipartite graphs.

References

[1] W. Badreddine, N. Khernane, M. Potop-Butucaru, and C. Chaudet. Convergecast in wireless body area networks. *Ad Hoc Networks*, 66:40–51, 2017.

[2] D. Bal, P. Bennett, A. Dudek, and P. Pralat. The total acquisition number of random graphs. *Electronic Journal of Combinatorics*, 23(2):P2.55, 2016.

[3] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[4] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

[5] G. Bu and M. Potop-Butucaru. Fifo order reliable convergecast in wban. *Computer Networks*, 146:200–216, 2018.

[6] S. Gandham, Y. Zhang, and Q. Huang. Distributed time-optimal scheduling for convergecast in wireless sensor networks. *Computer Networks*, 52(3):610–629, 2008.

[7] M. R. Garey and D. S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

[8] A. Godbole, E. Kelley, E. Kurtz, P. Pralat, and Y. Zhang. The total acquisition number of the randomly weighted path. *Discussiones Mathematicae Graph Theory*, 37:919–934, 2017.

[9] E. Infeld, , D. Mitsche, and P. Pralat. The total acquisition number of random geometric graphs. *Electronic Journal of Combinatorics*, 24(3):P3.31, 2017.

[10] R. Jiang, Y. Zhu, and Y. Yang. Improving throughput and fairness of convergecast in vehicular networks. *IEEE Transactions on Mobile Computing*, 16(11):3070–3083, 2017.

[11] F. Johnson, A. Raleigh, P. S. Wenger, and D. B. West. The unit acquisition number of a graph. *Discrete Applied Mathematics*, 258:166–176, 2019.

[12] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, New York, 1972. Plenum Press.

[13] A. Kesselman and D. R. Kowalski. Fast distributed algorithm for convergecast in ad hoc geometric radio networks. *Journal of Parallel and Distributed Computing*, 66(4):578–585, 2006.

[14] D. E. Lampert and P. J. Slater. The acquisition number of a graph. *Congressus Numerantium*, 109:203–210, 1995.

[15] T. D. Lesaulnier, N. Prince, P. S. Wenger, D. B. West, and P. Worah. Total acquisition in graphs. *SIAM Journal on Discrete Mathematics*, 27(4):1800–1819, 2013.

[16] T. D. Lesaulnier and D. B. West. Acquisition-extremal graphs. *Discrete Applied Mathematics*, 161:1521–1529, 2013.

[17] L. MacDonald, P. S. Wenger, and S. Wright. Total acquisition on grids. *Australasian Journal of Combinatorics*, 58(1):137–156, 2014.

[18] B. Malhotra, I. Nikolaidis, and M. A. Nascimento. Aggregation convergecast scheduling in wireless sensor networks. *Wireless Networks*, 17(2):319–335, 2011.

[19] H. Shi, M. Zheng, W. Liang, and J. Zhang. Convergecast scheduling for industrial wireless sensor networks with local available channel sets. *IEEE Sensors Journal*, 19(22):10764–10772, 2019.

[20] P. J. Slater and Y. Wang. The competitive-acquisition numbers of paths. *Congressus Numerantium*, 167:33–43, 2004.

[21] P. J. Slater and Y. Wang. Some results on acquisition numbers. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 64:65–78, 2008.

[22] P. S. Wenger. Fractional acquisition in graphs. *Discrete Applied Mathematics*, 178:142–148, 2014.