PATHWIDTH is NP-hard for weighted trees

Rodica Mihai[1] and Ioan Todinca[2]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway,
Rodica.Mihai@ii.uib.no
[2] LIFO, Université d'Orléans, 45067 Orléans Cedex 2, France,
Ioan.Todinca@univ-orleans.fr

Abstract. The pathwidth of a graph G is the minimum clique number of H minus one, over all interval supergraphs H of G. We prove in this paper that the PATHWIDTH problem is NP-hard for particular subclasses of chordal graphs, and we deduce that the problem remains hard for weighted trees. We also discuss subclasses of chordal graphs for which the problem is polynomial.

1 Introduction

Graph searching problems, already introduced in the 60's [5], have become popular in computer science with the seminal papers of Parsons [24, 23] and of Petrov [27] and have been widely studied since the late 70's. In graph searching we are given a contaminated graph (all edges are contaminated) and we aim to clean it using a minimum number of searchers. There are several variants of the problem, let us consider here the *node searching* introduced by Kirousis and Papadimitriou [16]. A search step consists in one of the following operations: (1) place a searcher on a vertex, (2) remove a searcher from a vertex. A contaminated edge becomes clear if both its endpoints contain a searcher. A clear edge is recontaminated at some step if there exists a path from this edge and a contaminated edge, and no internal vertex of this path contains a searcher. The *node search number* of a graph is the minimum number of searchers required to clean the whole graph, using a sequence of search steps. By the results of LaPaugh [19] and Bienstock and Seymour [2], the node search game is monotone: there always exists a cleaning strategy, using a minimum number of searchers, such that a clear edge will never be recontaminated during the search process.

The *vertex separation number* of a graph is defined as follows. A *layout* of a graph G is a total ordering $(v_1, v_2, \ldots, v_n)$ on its vertex set. The separation number of a given layout is the maximum, over all positions $i, 1 \leq i \leq n$, of the number of vertices in $\{v_1, \ldots, v_i\}$ having neighbours in $\{v_{i+1}, \ldots, v_n\}$. (Informally, one can think that at step i, the part of the graph induced by the first i vertices has been cleaned, the rest is contaminated, and we need to guard all clean vertices having contaminated neighbours in order to avoid recontamination.) The *vertex separation number* of a graph G is the minimum separation number over all possible layouts of G.

Pathwidth has been introduced in the first article of Robertson and Seymour's Graph Minor series [28]. The pathwidth of an arbitrary graph G is the minimum clique number of H minus one, over all interval supergraphs H of G, on the same vertex set.

Actually, the three graph parameters mentioned above are almost the same. For any graph G, its pathwidth $\mathrm{pwd}(G)$ is exactly the vertex separation number, and is also equal to the node search number minus one, [15]. From now on we consider the PATHWIDTH problem, i.e. the problem of computing the pathwidth of the input graph.

The PATHWIDTH problem is NP-hard, even for co-bipartite graphs [32] and chordal graphs [11]. On the positive side, it is fixed parameter tractable: Bodlaender and Kloks [3] proposed an algorithm that, given an arbitrary graph G and a parameter k, the algorithm decides if $\mathrm{pwd}(G) \leq k$ in a running time which is linear in n (the number of vertices of the graph) but, of course, exponential in k. By [3], PATHWIDTH is polynomially tractable for all classes of graphs of bounded treewidth. For trees and forests there exist several algorithms solving the problem in $\mathcal{O}(n \log n)$ time [20, 8]; recently, Skodinis [29] and Peng et al. [25] gave a linear time algorithm. Even the unicyclic graphs, obtained from a tree by adding one edge, require more complicated algorithms in order to obtain an $\mathcal{O}(n \log n)$ time bound [7]. Chou *et al.* [6] extended the techniques used on trees and obtain a polynomial algorithm solving the problem on block graphs, i.e. graphs in which each 2-connected component induces a clique. There exist some other geometric graph

classes for which the PATHWIDTH problem is polynomial, for example permutation graphs [4]. Based on this result, Peng and Yang [26] solved the problem for biconvex bipartite graphs.

Suchan and Todinca [30] gave an $\mathcal{O}(n^2)$ computing the pathwidth of circular-arc graphs; circular arc-graphs are intersection graphs of a family of arcs of a cycle (see next section for more details on intersection graphs). The technique used in [30] exploits the geometric structure of circular arc graphs and the fact that such a graph can be naturally cut using some sets of scanlines (chords of the cycle). A similar approach has been used before for computing the treewidth of circular-arc graphs [31] and the treewidth of circle graphs [17]. Circle graphs are intersection graphs of a set of chords of a cycle, i.e. each vertex of the graph corresponds to a chord and two vertices are adjacent if the corresponding chords intersect. The class of circle graphs contains the class of distance-hereditary graphs and in particular all the trees [10]. The problem of computing pathwidth for distance heredetary (and thus circle graphs) was proved to be NP-hard [18].

Our results. We prove in this paper that PATHWIDTH remains NP-hard even when restricted to weighted trees with polynomial weights. The weighted version of pathwidth is defined in the next section; roughly speaking, in terms of search or vertex separation, when we guard a vertex v of weight $w(v)$, we need $w(v)$ guards instead of one. Equivalently, the pathwidth of a weighted tree can be seen as the pathwidth of the unweighed graph obtained by replacing each vertex v of the tree with a clique module of size $w(v)$. Since the latter graphs are distance-hereditary, this also implies the NP-hardness of pathwidth for distance-hereditary and circle graphs. We also show that MODIFIED CUTWIDTH is NP-hard to compute for edge-weighted trees. Note that Monien and Sudborough [21] proved that CUTWIDTH is NP-hard for edge-weighted trees, and our reductions are inspired by their techniques. Eventually, we discuss some classes of graphs for which the PATHWIDTH problem remains polynomial.

2 Preliminaries

2.1 Basic definitions

We work with simple and undirected graphs $G = (V, E)$, with vertex set $V(G) = V$ and edge set $E(G) = E$, and we let $n = |V|$, $m = |E|$. The set of *neighbors* of a vertex x is denoted by $N(x) = \{y \mid xy \in E\}$. A vertex set C is a *clique* if every two vertices in C are adjacent, and a *maximal clique* if no superset of C is a clique. We denote by $\omega(G)$ the maximum clique size of the graph. A set of vertices M of G forms a *module* if, for any vertex $x \in V \setminus M$, either x is adjacent to all vertices of M or to none of them. The subgraph of G induced by a vertex set $A \subseteq V$ is denoted by $G[A]$. A *path* is a sequence $v_1, v_2, ..., v_p$ of distinct vertices of G, where $v_i v_{i+1} \in E$ for $1 \leq i < p$, in which case we say that this is a path *between* v_1 and v_p. A path $v_1, v_2, ..., v_p$ is called a *cycle* if $v_1 v_p \in E$. A *chord* of a cycle (path) is an edge connecting two non-consecutive vertices of the cycle (path). A vertex set $S \subset V$ is a *separator* if $G[V \setminus S]$ is disconnected. Given two vertices u and v, S is a u, v*-separator* if u and v belong to different connected components of $G[V \setminus S]$, and S is then said to *separate* u and v. A u, v-separator S is *minimal* if no proper subset of S separates u and v. In general, S is a *minimal separator* of G if there exist two vertices u and v in G such that S is a minimal u, v-separator.

A graph is *chordal* if every cycle of length at least 4 has a chord. Given a family $\mathcal{F}$ of sets, the intersection graph of the family is defined as follows. The vertices of the graph are in one-to-one correspondence with the sets of $\mathcal{F}$, and two vertices are adjacent if and only if the corresponding sets intersect. Every graph is the intersection graphs of some family $\mathcal{F}$, but by restricting these families we obtain interesting graph classes.

An *interval graph* is the intersection graph of a family of intervals of the real line. A graph is *circle graph* if it is the intersection graph of chords in a circle.

Definition 1. *A* path decomposition *of an arbitrary graph $G = (V, E)$ is a path $\mathcal{P} = (\mathcal{X}, A)$, where the nodes $\mathcal{X}$ are subsets of V (also called* bags*), such that the following three conditions are satisfied.*

1. *Each vertex $v \in V$ appears in some bag.*
2. *For every edge $\{v, w\} \in E$ there is a bag containing both v and w.*

3. *For every vertex $v \in V$, the bags containing v induce a connected subpath of $\mathcal{P}$.*

The width of the path decomposition $\mathcal{P} = (\mathcal{X}, A)$ is $\max\{|X| - 1 \mid X \in \mathcal{X}\}$ (the size of the largest bag, minus one). The pathwidth *of G, denoted $\mathrm{pwd}(G)$, is the minimum width over all path decompositions of the graphs.*

We extend the definition of pathwidth to weighted graphs $G = (V, E, w : V \to \mathbb{N})$, where the weights are assigned to the vertices. The weight of a path decomposition $\mathcal{P} = (\mathcal{X}, A)$ is $\max\{w(X) - 1 \mid X \in \mathcal{X}\}$, where $w(X)$ is the weight of the bag, i.e. the sum of the weights of its nodes. The (weighted) pathwidth is again the minimum weight over all path decompositions of the graph.

Observation 1 *The weighted pathwidth of a graph $G = (V, E, w)$ equals the (unweighted) pathwidth of the graph H, obtained from G by replacing each vertex v by a clique module M_v of size $w(v)$. That is, we replace each vertex v by set of vertices M_v, of size $w(v)$ and inducing a clique in H, and two vertices $v' \in M_v$ and $u' \in M_u$ are adjacent in H if and only if v and u are adjacent in G.*

Path decompositions are strongly related to interval graph. Given a graph G, a *clique path* of G is a path decomposition whose set of bags is exactly the set of maximal cliques of G. It is well known that a graph G is an interval graph if and only of it has a clique path. Simillarilly, a *clique tree* of a graph G is a tree whose nodes correspond to the maximal cliques of G, and such that for any vertex of the graph, the nodes containing this vertex form a connected subtree of the clique tree. Clique trees characterize chordal graphs:

Lemma 1 (see, e.g., [9]). *A graph G is an interval graph if and only of it has a clique path. A graph G is chordal if and only if it has a clique tree.*

Clearly, clique paths are particular cases of clique trees, in particular interval graphs are also chordal.

An interval completion of an arbitrary graph $G = (V, E)$ is an interval supergraph $H = (V, F)$ of G, with the same vertex set and $E \subseteq F$. Moreover, if no strict subgraph H' of H is an interval completion of G, we say that H is a *minimal interval completion* of G. By the previous lemma, the pathwidth of an (unweighted) interval graph is its clique size minus one, and the pathwidth of an arbitrary unweighted graph is the maximum, over all interval completions H of G, of $\omega(H) - 1$. Moreover, when searching for interval completions realizing this minimum we can clearly restrict to minimal interval completions.

2.2 Foldings

Given a path decomposition $\mathcal{P}$ of G, let $\mathrm{PathFill}(G, \mathcal{P})$ be the graph obtained by adding edges to G so that each bag of $\mathcal{P}$ becomes a clique. It is straight forward to verify that $\mathrm{PathFill}(G, \mathcal{P})$ is an interval supergraph of G, for every path decomposition $\mathcal{P}$. Moreover $\mathcal{P}$ is a path decomposition of $\mathrm{PathFill}(G, \mathcal{P})$, and if we remove the bags which are not maximal by inclusion we obtain a clique path.

Definition 2. *Let $\mathcal{X} = \{X_1, ..., X_k\}$ be a set of subsets of V such that X_i, $1 \leq i \leq k$, is a clique in $G = (V, E)$. If, for every edge $v_i v_j \in E$, there is some X_p such that $\{v_i, v_j\} \subseteq X_p$, then $\mathcal{X}$ is called an* edge clique cover *of G.*

Definition 3 ([13, 30]). *Let $\mathcal{X}$ be an edge clique cover of an arbitrary graph G and let $\mathcal{Q} = (Q_1, ..., Q_k)$ be a permutation of $\mathcal{X}$. We say that $(G, \mathcal{Q})$ is a* folding *of G by $\mathcal{Q}$.*

To any folding of G by an ordered edge clique cover $\mathcal{Q}$ we can naturally associate, by Algorithm $\mathrm{FillFolding}$ of Figure 1, an interval supergraph $H = \mathrm{FillFolding}(G, \mathcal{Q})$ of G. The algorithm also constructs a clique path decomposition of H.

Lemma 2 ([30]). *Given a folding $(G, \mathcal{Q})$ of G, the graph $H = \mathrm{FillFolding}(G, \mathcal{Q})$ is an interval completion of G.*

We also say that the graph $H = \mathrm{FillFolding}(G, \mathcal{Q})$ is *defined* by the folding $(G, \mathcal{Q})$. It is not hard to prove (see [30]) that every minimal interval completion of G is defined by some folding.

Algorithm FillFolding
Input: Graph $G = (V, E)$ and $\mathcal{Q} = (Q_1, ..., Q_k)$, a sequence of subsets of V forming an edge-clique cover;
Output: A supergraph H of G;

$\mathcal{P} = \mathcal{Q}$;
for each vertex v of G **do**
  $s = \min\{i \mid x \in Q_i\}$;
  $t = \max\{i \mid x \in Q_i\}$;
  for $j = s + 1$ to $t - 1$ **do**
    $P_j = P_j \cup \{v\}$;
end-for
$H = \text{PathFill}(G, \mathcal{P})$;

Fig. 1. The FillFolding algorithm.

Theorem 1 ([30]). *Let H be a minimal interval completion of a graph G with an edge clique cover $\mathcal{X}$. Then there exists a folding $(G, \mathcal{Q}_H)$, where $\mathcal{Q}_H$ is a permutation of $\mathcal{X}$, such that $H = \text{FillFolding}(G, \mathcal{Q}_H)$. In particular, there exists a folding $\mathcal{Q}$ such that the pathwidth of $\text{FillFolding}(G, \mathcal{Q})$ is exactly the pathwidth of G.*

For any chordal graph G, the set $\mathcal{K}$ of its maximal cliques form an edge-clique cover the graph, thus we will construct foldings of G by choosing permutations of $\mathcal{K}$. The next two lemmas give a better understanding of such a folding.

Lemma 3 (see, e.g., [9]). *Let G be a chordal graph and fix a clique tree of G. A vertex subset S of G is a minimal separator if and only if there exist two maximal cliques K and K' of G, adjacent in the clique tree, such that $S = K \cap K'$.*

Lemma 4. *Let G be a chordal graph, $\mathcal{K}$ be the set of its maximal cliques and let $\mathcal{Q}$ be a permutation of $\mathcal{K}$. Let T be a clique tree of G. Consider the path decomposition $\mathcal{P}$ produced by the algorithm $\text{FillFolding}(G, \mathcal{Q})$ (see Figure 1). Each bag B of $\mathcal{P}$ is the union the clique $Q \in \mathcal{Q}$ which corresponds to the bag B at the initialization step and of the minimal separators of type $Q' \cap Q''$, where Q', Q'' are the pairs of cliques adjacent in the clique tree, but separated by Q in $\mathcal{Q}$. We say that the clique Q and the separators have been* merged *by the folding.*

Proof. Clearly if Q separates Q' and Q'' in the permutation $\mathcal{Q}$, by construction of $\text{FillFolding}(G, \mathcal{Q})$, we add to bag B every vertex in $Q' \cap Q''$.

Conversely, let B be the bag corresponding to $Q \in \mathcal{Q}$ in $\text{FillFolding}(G, \mathcal{Q})$ and let x be a vertex of $B \setminus Q$. We have to prove there exist two maximal cliques of G, adjacent in the clique tree, containing x and separated by Q in the permutation $\mathcal{Q}$. By definition of a clique tree, the nodes of T containing x induce a connected subtree T_x. Let Q_l, Q_r be maximal cliques containing x, situated to the left and to the right of Q in $\mathcal{Q}$ (they exist by the fact that x has been added to bag B). Thus Q_l and Q_r are nodes of T_x, and on the unique path from Q_l to Q_r in T_x there exists an edge $Q'Q''$ such that Q' is to the left and Q'' is to the right of Q in the permutation $\mathcal{Q}$. □

3 Octopus graphs

3.1 Octopus graphs and 0,1-linear equations

An *octopus tree* is a tree formed by several disjoint paths, plus a node, called the head of the octopus, adjacent to one endpoint of each path. An emphoctopus graph is a chordal graph having a clique-tree which is an octopus tree, and such that any vertex of the graph appears in at most two maximal cliques. The last condition implies, by Lemma 3, that the minimal separators of an octopus graph are pairwise disjoint.

Consider the following problem, called the solvability problem for linear integer equations:

– *Input:* A rectangular matrix A and a column vector b, both of nonnegative integers.
– *Output:* Does there exist a 0-1 column vector x satisfying $Ax = b$?

We will use a restricted version of this problem, that we call the BALANCED SYSTEM OF 0,1-LINEAR EQUATIONS, satisfying the following conditions:

1. for all rows i of matrix A, the sum of the elements in the row is exactly $2b_i$,
2. for all rows i, all values in row i are larger than all values in row $i + 1$.

Note that when matrix A has only one row we obtain the 2-partition problem. Monien and Sudborough prove in [21] that:

Theorem 2 ([21]). *Solving balanced 0,1-linear equation systems is strongly NP-hard, i.e. the problem is hard even when the integers in the matrices A and b are polynomially bounded by the size of A.*

Monien and Sudborough use a version of this problem for a reduction to CUTWIDTH of edge-weighted trees, which shows that the latter is NP-hard. Our reductions are also from BALANCED SYSTEM OF 0,1-LINEAR EQUATIONS, inspired by their reduction. Nevertheless, it is not clear that one could directly find a reduction from CUTWIDTH of edge-weighted trees to PATHWIDTH of (node) weighted trees or circle graphs or other problems we consider in this paper.

3.2 NP-hardness of PATHWIDTH for octopus graphs

This subsection is devoted to the proof of the following theorem:

Theorem 3. PATHWIDTH *is NP-hard for octopus graphs.*

More technically, we prove by the following Proposition that the BALANCED SYSTEM OF 0,1-LINEAR EQUATIONS problem is polynomially reducible to PATHWIDTH of octopus graphs.

Proposition 1. *Given an instance (A, x) of* BALANCED SYSTEM OF 0,1-LINEAR EQUATIONS, *we construct an octopus graph $G(A, b)$ as follows:*

– *$G(A, b)$ has a clique tree formed by a head and n legs with m nodes each, where n is the number of columns and m is the number of rows of A.*
– *Let K_H be the clique of $G(A, b)$ corresponding to the head of the octopus, and for $j, 1 \leq j \leq n$ and each $i, 1 \leq i \leq m$ let $K_{i,j}$ denote the clique corresponding to the node of the jth leg, situated at distance i from the head.*
  - *the head clique K_H is of size $b_1 + k$, where k is an arbitrary number larger than b_1;*
  - *for each i, j, $K_{i,j}$ is of size $A_{i,j} + b_1 - b_i + k$;*
  - *for each i, j, the minimal separator $S_{i,j} = K_{i-1,j} \cap K_{i,j}$ is of size $A_{i,j}$; here $K_{0,j}$ denotes the head clique K_H.*

$$A = \begin{bmatrix} 15 & 15 & 15 & 15 \\ 10 & 10 & 10 & 10 \\ 5 & 5 & 5 & 5 \end{bmatrix} \quad B = \begin{bmatrix} 30 \\ 20 \\ 10 \end{bmatrix}$$

Fig. 2. Matrices

We have $\mathrm{pwd}(G(A, b)) \leq b_1 + k - 1$ if and only if the system $Ax = b$ has a 0,1-solution.

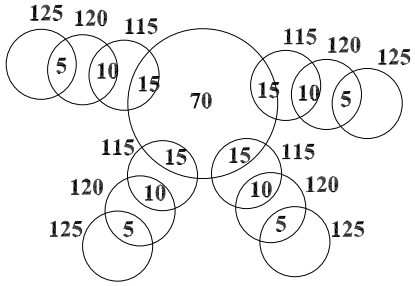

Fig. 3. Octopus graph corresponding to the matrices from Figure 2

Proof. For simplicity, the graph $G(A, b)$ will be simply denoted by G. Recall that the minimal separators of G are pairwise disjoint and each vertex of the graph appears in at most two maximal cliques. We point out that, by our choice of $k \geq b_1$, and by the fact that $b_1 \geq b_i$ for all $i > 1$, the graph G is well constructed.

We first show that, if the system $Ax = b$ has a 0,1-solution then $\mathrm{pwd}(G) \leq b_1 + k - 1$ (actually it is easy to notice that the inequality can be replaced with an equality, due the fact that G has a clique of size $b_1 + k$). Suppose the system has a solution x^*. We provide a folding of G by putting all legs j of the octopus such that $x^*_j = 0$ to the left, and all those with $x^*_j = 1$ to the right of the head clique K_H. Moreover, the folding is such that, on each side of the the head clique K_H, starting from K_H, we first put the maximal cliques at distance 1 from the head in the clique tree, then those at distance 2 and so on, like in a breadth-first search of the clique tree starting from the head.

More formally, let $L = \{j, 1 \leq j \leq n \mid x^*_j = 0\}$ and $R = \{j, 1 \leq j \leq n \mid x^*_j = 1\}$. We construct a folding $\mathcal{Q}$ as follows. The head clique K_H is in the middle position of the folding. We put to the right of K_H the cliques $K_{i,j}$, for all $j \in R$ and all $i, 1 \leq i \leq m$. The ordering is such that for any $i_1 \leq i_2$, and any j_1, j_2, K_{i_1,j_1} appears before K_{i_2,j_2} in $\mathcal{Q}$. We may assume that, for a fixed i, the cliques $K_{i,j}, j \in R$ appear by increasing j. The cliques appearing before K_H in the permutation are ordered symmetrically w.r.t. K_H. Let $H = \mathrm{FillFolding}(G, \mathcal{Q})$ and $\mathcal{P}$ the path decomposition corresponding to the folding, it remains to prove that $\mathcal{P}$ has no bag with more than $b_1 + k$ vertices. Since each leg of the octopus is either completely to the left or completely to the right of the clique K_H, this clique does not separate, in the permutation $\mathcal{Q}$, any couple of cliques adjacent in the clique tree. Therefore, by Lemma 4, the bag corresponding to K_H in $\mathcal{P}$ has no other vertices than K_H. Consider now a bag of $\mathcal{P}$, corresponding to a clique $K_{i,j}$. We assume w.l.o.g. that $j \in R$, the case $j \in L$ is similar by symmetry of the folding. This bag, denoted by $B_{i,j}$, contains $K_{i,j}$ and the vertices of all the separators corresponding to edges of the clique tree, such that $K_{i,j}$ separates the endpoints of the edge in the permutation $\mathcal{Q}$ (see Lemma 4). These separators are of two types: $S_{i,j'}$ with $j' \in R, j' > j$ and $S_{i+1,j''}$ with $j'' \in R, j'' < j$. Their sizes are respectively $A_{i,j'}$ and $A_{i+1,j''}$, and by definition of balanced systems $A_{i+1,j''} \leq A_{i,j''}$. Since $|K_{i,j}| = A_{i,j} + b_1 - b_i + k$, the size of the bag $B_{i,j}$ is at most $b_1 - b_i + k + \sum_{j \in R} A_{i,j}$ Note that, for any fixed i the equality is attained by the bag B_{i,j_0} which is closest to K_H in the permutation. By definition of set R, the sum is exactly b_i and the conclusion follows.

Now conversely, assume that $\mathrm{pwd}(G) \leq b_1 + k - 1$, we show that the system $Ax = b$ has a 0,1-solution. Let $\mathcal{Q}$ be a permutation of the maximal cliques of G such that $\mathrm{FillFolding}(G, \mathcal{Q})$ is of pahwidth at most $b_1 + k - 1$; such a folding exists by Theorem 1.

Claim. For any leg j of the octopus tree, all cliques $K_{i,j}$, $1 \leq i \leq m$ of the leg are on the same side of the head clique K_H in the permutation $\mathcal{Q}$.

Proof. (of the claim) Indeed if K_H separates in the permutation $\mathcal{Q}$ two cliques of a same leg j, then it necessarily separates two consecutive cliques $K_{i,j}$ and $K_{i+1,j}$, for some $i, 1 \leq i < m$. It follows by Lemma 4 that, when constructing $\mathrm{FillFolding}(G, \mathcal{Q})$, the bag of the resulting path decomposition, corresponding to the clique K_H, contains both K_H and the minimal separator $S_{i,j} = K_{i,j} \cap K_{i+1,j}$. Since K_H and $S_{i,j}$ are disjoint by construction of G, this bag would have strictly more than $|K_H| = b_1 + k$

vertices, contradicting the assumption that the pathwidth of $\mathrm{FillFolding}(G, \mathcal{Q})$ is at most $b_1 + k - 1$. This completes the proof of the claim. □

Let L be the set of indices $j, 1 \leq j \leq n$ such that the cliques of leg j appear before K_H in $\mathcal{Q}$, and symmetrically let R be the set of indices corresponding to legs appearing after K_H.

Claim. For all i, $1 \leq i \leq m$, we have

$$\sum_{j \in L} A_{i,j} = \sum_{j \in R} A_{i,j} = b_i.$$

Proof. (of the claim) Take an arbitrary value of $i, 1 \leq i \leq m$. We prove that $\sum_{j \in R} A_{i,j} \leq b_i$. Among all cliques $K_{i,j}$, $j \in R$ let j_0 be the index j corresponding to the one which is closest to K_H in the ordering $\mathcal{Q}$. Therefore, for any $j \in R \setminus \{j_0\}$, there are two cliques $K_{i'-1,j}$ and $K_{i',j}$, with $i' \leq i$, separated by K_{i,j_0} in the permutation $\mathcal{Q}$ (here again we use the convention $K_{0,j} = K_H$). In $\mathrm{FillFolding}(G, \mathcal{Q})$, the separator $S_{i',j}$ is merged to the clique K_{i,j_0}, adding $A_{i',j} \geq A_{i,j}$ new vertices in the corresponding bag. Thus this bag will have at least $|K_{i,j_0}| + \sum_{j \in R \setminus \{j_0\}} A_{i,j}$ vertices, so its size is at least $b_1 - b_i + k + \sum_{j \in R} A_{i,j}$. Since the pathwidth of $\mathrm{FillFolding}(G, \mathcal{Q})$ is at most $b_1 + k - 1$, each bag produced by the folding is of size at most $b_1 + k$, in particular we must have $\sum_{j \in R} A_{i,j} \leq b_i$. By symmetry, we also have $\sum_{j \in R} A_{i,j} \leq b_i$, and since the sum of all elements in row i is $2b_i$ the conclusion follows. □

By the last claim, the 0,1-column vector x^* such that, for all $j, 1 \leq j \leq n$, $x^*_j = 0$ if $j \in L$ and $x^*_j = 1$ if $j \in L$ is a solution of the system $Ax = b$, which completes the proof of this Proposition. □

Clearly the graph $G(A, b)$ can be constructed in polynomial time, thus the BALANCED SYSTEM OF 0,1-LINEAR EQUATIONS problem is polynomially reducible to PATHWIDTH of octopus graphs. By Theorem 2, we conclude that PATHWIDTH is NP-hard even when restricted to octopus graphs, which proofs Theorem 3.

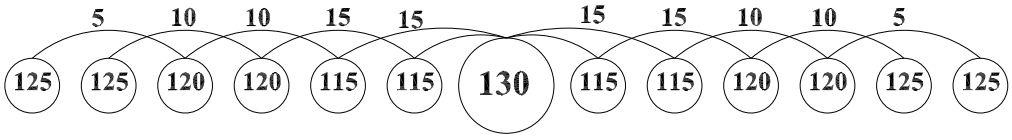

Fig. 4. Folding for the octopus graph from Figure 3

4 Weighted trees

We prove in this section that PATHWIDTH is hard for weighted trees.

Let us consider now the case of weighted trees. We adapt the ideea of Proposition 1 by transforming a system of 0,1-linear equation into a pathwidth problem for weighted trees.

Proposition 2. *Given an instance (A, x) of* BALANCED SYSTEM OF 0,1-LINEAR EQUATIONS*, we construct an weighted octopus tree $T(A, b)$ as follows:*

- *$T(A, b)$ is formed by a head, n legs with $2m$ nodes each, where n is the number of columns and m is the number of rows of A, plus an node adjacent only to the head.*
- *Let N be the head node of the tree. The nodes of leg j are denoted $S_{1,j}, C_{1,j}, S_{2,j}, C_{2,j}, \ldots, S_{m,j}, C_{m,j}$ and appear in this order; $S_{1,j}$ is the one adjacent to the head. (The S-nodes play the same role as the minimal separators for octopus graphs.)*

- *the head N is of weight k, where k is an arbitrary number larger than $2b_1$;*
- *the node N' adjacent only to N is of weight b_1;*
- *for each i, j, the node $C_{i,j}$ is of weight $b_1 - b_i + k$;*
- *for each i, j, the node $S_{i,j}$ is of weight $A_{i,j}$.*

We have $\mathrm{pwd}(T(A,b)) \leq b_1 + k - 1$ *if and only if the system $Ax = b$ has a 0,1-solution.*

Proof. Let $H(A,b)$ the graph obtained by replacing, in $T(A,b)$, each node by a clique module such that the number of vertices of the clique is exactly the weight of the node like in Observation 1. We have $\mathrm{pwd}(T(A,b)) = \mathrm{pwd}(H(A,b))$. Note that $H(A,b)$ is chordal. The maximal cliques of $H(A,b)$ correspond now to edges of $T(A,b)$, i.e. the maximal cliques are exactly the unions of two clique modules corresponding to adjacent nodes of T. Let us use the same notations for the clique modules of $H(A,b)$ as for the corresponding vertices of $T(A,b)$. Although $H(A,b)$ is not exactly an octopus graph, because the vertices of N appear in several maximal cliques, its structure is quite similar. In particular it has a clique tree such that the corresponding tree, denoted $\tilde{T}$, is an octopus. The head of the octopus tree is the clique $N \cup N'$. Each leg of $\tilde{T}$ corresponds to one of the n long paths of $T(A,b)$. The cliques of leg j are, starting from the head, the cliques corresponding to the $2m$ edges of the jth path from N to the leaf $S_{n,j}$: $N \cup S_{1,j}, S_{1,j} \cup C_{1,j}, C_{1,j} \cup S_{2,j}, S_{2,j} \cup C_{2,j}, \ldots, S_{n,j} \cup C_{n,j}$.

Assume that $\mathrm{pwd}(H(A,b)) \leq b_1 + k - 1$, we prove that the system $Ax = b$ has a 0,1-solution. For this purpose we use the octopus chordal graph $G(A,b)$ constructed in Proposition 1, and it is sufficient to show that $\mathrm{pwd}(G(A,b)) \leq \mathrm{pwd}(H(A,b))$.

Recall that a graph G_1 is a *minor* of a graph G_2 if G_1 can be obtained from G_2 by a series of edge deletions, vertex deletions and edge contractions (contracting an edge uv consists in replacing this edge by a single vertex, whose neighborhood is the union of the neighborhoods of u and of v). It is well-known that, if G_1 is a minor of G_2, then the pathwidth of G_1 is at most the pathwidth of G_2 [28].

Claim. $G(A,b)$ is a minor $H(A,b)$.

Proof. (of the claim) In $H(A,b)$, for each j, $1 \leq j \leq m$, let $S'_{i-1,j}$ be a subset of $C_{i-1,j}$, of size $A_{i,j}$. Here $C_{0,j}$ denotes N, and we ensure that the sets $S_{0,1}, S_{0,2}, \ldots, S_{0,j}$ are pairwise disjoint, which is possible by our choice of $k \geq 2b_1 = \sum_{j=1}^{n} A_{1,j}$. Choose a matching between $S_{i,j}$ and $S'_{i-1,j}$, for each pair i, j, and contract each edge of the matching ; the set of vertices obtained by these contractions is denoted $U_{i,j}$. We claim that the new graph is exactly $G(A,b)$. Note that the edges of all these matchings are pairwise disjoint. In the octopus clique tree $\tilde{T}$ of $H(A,b)$, let us consider the jth leg $C_{0,j} \cup S_{1,j}, S_{1,j} \cup C_{1,j}, C_{1,j} \cup S_{2,j}, S_{2,j} \cup C_{2,j}, \ldots, S_{n,j} \cup C_{n,j}$. The cliques of type $C_{i-1,j} \cup S_{i,j}$ are smaller than the cliques $C_{i,j} \cup S_{i,j}$. In particular, after applying our contractions, in the new graph the leg has been transformed into the following sequence of maximal cliques $N, U_{1,j} \cup C_{1,j}, U_{2,j} \cup C_{2,j}, \ldots, U_{n,j} \cup C_{n,j}$ (N is not a maximal clique of the new graph but only of its restriction to the leg). It is sufficient to notice that the size of the clique $U_{i,j} \cup C_{i,j}$ is $A_{i,j} + b_1 - b_i + k$, and intersection of two consecutive cliques corresponds to sets $U_{i,j}$, of size $A_{i,j}$. The new graph is indeed $G(A,b)$. □

Conversely, assume that the system $Ax = b$ has a 0,1-solution x^*, we give a folding of $H(A,b)$ producing a path decomposition of width $b_1 + k - 1$. Like in the proof of Proposition 1, let R (resp. L) be the set of indices i, $1 \leq j \leq n$ such that $x^*_j = 1$ (resp. $x^*_j = 0$). We only discuss the folding to the right of the clique $N \cup N'$, the left-hand side being symmetrical. To the right of $N \cup N'$ we put the cliques $N \cup S_{1,j}$ for each $j \in R$. Then,

for each i, $1 \leq i \leq m$ in increasing order,
  for each $j \in R$
    we append at the end the cliques $S_{i,j} \cup C_{i,j}$ and $C_{i,j} \cup S_{i+1,j}$

It remains to prove that the width of the corresponding folding is at most $b_1 + k - 1$. The bags corresponding to cliques of type $N \cup S_{1,j}$ wil be merged with all other separators $S_{1,j'}$, $j' \in R \setminus \{j\}$, so they become of size $b_1 + k$. Among the other bags, those that give the width of the folding are the bags corresponding to cliques of type $S_{i,j} \cup C_{i,j}$. Indeed the bags corresponding to $C_{i,j} \cup S_{i+1,j}$ are merged in the folding with the same separators as the bag of $S_{i,j} \cup C_{i,j}$, and $|C_{i,j} \cup S_{i+1,j}| \leq |S_{i,j} \cup C_{i,j}|$, thus after the folding the former bags remain smaller than the latter. To each bag of these bags $B_{i,j}$ corresponding to $S_{i,j} \cup C_{i,j}$, we

add all other separators $S_{i',j'}$ with $j' \in R \setminus \{j\}$ and i' being either i or $i+1$. Thus, after the folding, the size of each of these bags is at most $|C_{i,j}| + \sum_{j' \in R} |S_{i,j}|$, which is precisely $k + b_1$. □

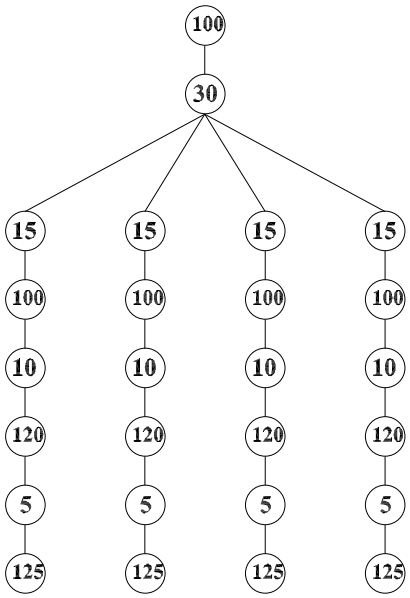

Fig. 5. Weighted tree corresponding to the matrices from Figure 2

The construction of the octopus tree $T(A, b)$ of Proposition 2 is clearly polynomial in the size of matrix A. By Theorem 2 we deduce:

Theorem 4. *The* PATHWIDTH *of weighted trees is NP-hard, even when the weights are polynomially bounded by the size of the tree.*

As a byproduct, this also implies a result already proved in [**?**], namely that PATHWIDTH remains NP-hard for distance-hereditary graphs and circle graphs. A graph is distance-hereditary if and only if it can be obtained starting from a unique vertex and repeatedely adding a new pendant vertices (with only one new neighbour), or vertices which are true twins or false twins of already existing vertices [**?**]. Recall that two vertices x and y are false twins (true twins) if $N(x) = N(y)$ ($N(x) \cup \{x\} = N(y) \cup \{y\}$. In particular all trees are distance-hereditary, and by adding true twins we obtain that the graphs obtained from weighted trees by replacing each vertex with a clique module are also distance-hereditary. Therefore Theorem 4 implies the NP-hardness of PATHWIDTH when restricted to distance hereditary graphs. Eventually, note that circle graphs contain all distance-hereditary graphs [**?**].

We can also prove that the MODIFIED CUTWIDTH problem is NP-hard on edge-weighted trees. The MODIFIED CUTWIDTH is a layout problem related to PATHWIDTH and CUTWIDTH. The proof of this result, although similar to the proofs of Theorems 3 and 4, is quite long and tedious, therefore the discussion about MODIFIED CUTWIDTH has been postponed to the Appendix.

5 PATHWIDTH of octopus graphs: polynomial cases

Pathwidth can be computed in polynomial time for chordal graphs having all minimal separators of size one, which are exactly the block graphs [6]. Already for chordal graphs with all minimal separators of size two it becomes NP-hard to compute pathwidth [11]. We proved that is NP-hard to compute pathwidth for octopus graphs, which also have a very simple structure since each vertex appears in at most two maximal

cliques. We will show how to compute pathwidth in polynomial time for a subclass of octopus graphs, namely octopus graphs with all separators of each leg of the same size.

Let G be an octopus graph with, for each leg j, all separators of the same size s_j.

Let G_1 be the graph obtained from G by removing from each leg all cliques except the maximum one. The clique tree of G_1 is a star and all separators are disjoint, the graph G_1 is a so-called primitive starlike graph. We can compute the pathwidth of G_1 using the algorithm for primitive starlike graphs of Gustedt [11].

Lemma 5. *There exists an ordering Q where the cliques of each leg appear consecutively such that the pathwidth of* $\mathrm{FillFolding}(G, \mathcal{Q})$ *is exactly the pathwidth of G.*

Proof. Let Q be an ordering for G such that the pathwidth of $\mathrm{FillFolding}(G, \mathcal{Q})$ is exactly the pathwidth of G. Ler $K_{l,j}$ be a clique of maximum size of the leg j. Let $\mathcal{Q}'$ be the folding obtained by (1) removing all other cliques of leg j and then (2) replacing them contigously to $K_{l,j}$, in the order of the leg ($K_{1,j}$ being the closest to the head). We claim that the new folding is also optimal. All cliques of leg j are merged with exactly the same separators as $K_{l,j}$, thus the size of their bags is upper bounded by the size of the bag corresponding to $K_{l,j}$. Let K be a clique which is not on the leg j. If K was merged before with one or more minimal separators of leg j, in the new folding it will be merged with at most one of these separators, so the corresponding bag does not increase. If K was not merged with a minimal separator of leg j, then in the new folding the bag of K remains unchanged. Therefore we do not modify the pathwidth of $\mathrm{FillFolding}(G, \mathcal{Q}')$.

Theorem 5. *The pathwidth of an octopus graph with, for each leg, all separators of the same size can be computed in polynomial time.*

Proof. Let G such an octopus graph. First we construct the graph G_1 by deleting from each leg of G all the cliques which are not maximum cliques. Let k be the maximum difference between the size of the head and the peripheral cliques. It follows that the graph G_1 is a primitive starlike graph [11]. Using the polynomial time algorithm of Gustedt we compute the pathwidth of G_1. In order to construct an ordering Q for G such that the pathwidth of $\mathrm{FillFolding}(G, \mathcal{Q})$ is exactly the pathwidth of G, we place next to each peripheral clique $K_{i,j}$ of G_1 all the cliques from leg j, respecting the ordering of the leg.

Clearly, when we have octopus graphs with a constant number of cliques the problem becomes polynomial, because the number of possible foldings is constant, too. It would be interesting to know if the PATHWIDTH problem on octopus graphs becomes polynomial when we restrict to octopuses with legs of constant size, or to octopuses with constant number of legs. In the latter case, using dynamic programming techniques, one can decide if the pathwidth is at most the size of the head of the octopus, minus one. More generally, if we know that there exists an optimal folding such that each leg is either completely to the left or completely to the right of the octopus, then we can also show using some results of [30] that the bags of a same leg should appear in the order of the leg. Then one can compute an optimal folding by dynamic programming, and the running time is of type $\mathcal{O}(Poly(n) \cdot n^l)$, where l is the number of legs of the octopus.

6 Conclusion

We have proved that the PATHWIDTH problem is NP-hard for weighted trees, even when restricted to polynomial weights, and also for the class of octopus graphs, which is a very restricted subset of chordal graphs. We have also shown that the MODIFIED CUTWIDTH problem is NP-hard for weighted trees.

Thus, despite the recent polynomial algorithms computing pathwidth for block graphs [6], biconvex bipartite graphs [26] or circular-arc graphs [30], the techniques for pathwidth computation do not seem extendable to large classes of graphs. A natural question is to search for good approximation algorithms for pathwidth. For chordal graphs and more generally graphs without long induced cycles, Nisse [22] proposed approximation algorithms whose approximation ratios do not depend of the size of the graph, but only on the pathwidth and the size of the longest induced cycle. It remains an open problem whether pathwidth can be approximated within a constant factor, even for the class of chordal graphs or the class of weighted trees.

Another interesting question is the CONNECTED SEARCH NUMBER on weighted trees. In the graph searching game, we say that a search strategy is connected if, at each step of the game, the clear area induces a connected subgraph [1]. The *connected search number* is the minimum number of searchers required to clean a graph using a connected strategy. This number can be computed in polynomial time for trees [1]. In the weighted version, we need $w(v)$ searchers to guard a vertex v of weight $w(v)$. The algorithm computing the connected search for unweighted trees cannot be straightforwardly extended to the weighted case. We should also point out that our NP-hardness proof for PATHWIDTH (and thus node search number) of weighted trees is strongly based on the fact that, in our class of octopus trees, the optimal search strategy is not connected, thus the proof cannot be immediately adapted to the connected search number problem.

References

1. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *Proceedings 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 200–209. ACM, 2002.
2. D. Bienstock and Paul Seymour. Monotonicity in graph searching. *J. Algorithms*, 12(2):239–245, 1991.
3. Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.
4. H.L. Bodlaender, T. Kloks, and D. Kratsch. Treewidth and pathwidth of permutation graphs. *SIAM J. on Discrete Math.*, 8:606–616, 1995.
5. R. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers*, VI(5):72–78, 1967.
6. Hsin-Hung Chou, Ming-Tat Ko, Chin-Wen Ho, and Gen-Huey Chen. Node-searching problem on block graphs. *Discrete Appl. Math.*, 156(1):55–75, 2008.
7. John Ellis and Minko Markov. Computing the vertex separation of unicyclic graphs. 192:123–161, 2004.
8. John A. Ellis, Ivan Hal Sudborough, and J. Turner. The vertex separation and search number of a graph. 113:50–79, 1994.
9. M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.
10. Agustín Gravano and Guillermo Durán. The intersection between some subclasses of circular-arc and circle graphs. *Congressus Numerantium*, 159:183–192, 2002.
11. J. Gustedt. On the pathwidth of chordal graphs. *Discrete Applied Mathematics*, 45(3):233–248, 2003.
12. M. Fellows H. Bodlaender and D. Thilikos. Starting with nondeterminism: the systematic derivation of linear-time graph layout algorithms. In *MFCS*, volume 2747, pages 239–248, 2003.
13. P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger. Characterizing minimal interval completions. In *STACS*, volume 4393 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 2007.
14. J., J. Petit, and M. J. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.
15. Nancy G. Kinnersley. The vertex separation number of a graph equals its path-width. *Inf. Process. Lett.*, 42(6):345–350, 1992.
16. M Kirousis and C H Papadimitriou. Searching and pebbling. *Theor. Comput. Sci.*, 47(2):205–218, 1986.
17. T. Kloks. Treewidth of circle graphs. *Intern. J. Found. Comput. Sci.*, 7:111–120, 1996.
18. T. Kloks, H.L. Bodlaender, H. Müller, and D. Kratsch. Computing treewidth and minimum fill-in: all you need are the minimal separators. In *Proceedings First Annual European Symposium on Algorithms (ESA'93)*, volume 726 of *Lecture Notes in Computer Science*, pages 260–271. Springer-Verlag, 1993.
19. A. S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40:224–245, 1993.
20. Nimrod Megiddo, S. L. Hakimi, Michael R. Garey, David S. Johnson, and Christos H. Papadimitriou. The complexity of searching a graph. *Journal of the ACM*, 35:18–44, 1988.
21. B. Monien and I. Sudborough. Min cut is np-complete for edge weighted treees. *Theoretical Computer Science*, 58:209–229, 1988.
22. Nicolas Nisse. Connected graph searching in chordal graphs. *Discrete Appl. Math.(in press)*.
23. T. Parsons. The search number of a connected graph. *Proceedings of the 9th Southeastern Conference on Combinatorics, Graph Theory, and Computing*.
24. T. Parsons. Pursuit-evasion in a graph. In *Theory and Applications of Graphs*. Springer-Verlag, 1976.
25. Sheng-Lung Peng, Chin-Wen Ho, Tsan-Shen Hsu, Ming-Tat Ko, and Chuan Yi Tang. Edge and node searching problems on trees. *Theoretical Computer Science*, 240:429–446, 2000.
26. Sheng-Lung Peng and Yi-Chuan Yang. On the treewidth and pathwidth of biconvex bipartite graphs. In *TAMC*, pages 244–255, 2007.
27. N. N. Petrov. A problem of pursuit in the absence of information on the pursued. *Differentsial nye Uravneniya*, 18:1345–1352, 1982.
28. N. Robertson and P. Seymour. Graph minors. I. Excluding a forest. *Journal of Combinatorial Theory Series B*, 35:39–61, 1983.

29. Konstantin Skodinis. Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time. 47:40–59, 2003.
30. K. Suchan and I. Todinca. Pathwidth of circular-arc graphs. In *33rd International Workshop on Graph-Theoretic Concepts in Computer Science, (WG 2007)*, volume 4769 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2007.
31. R. Sundaram, K. Sher Singh, and C. Pandu Rangan. Treewidth of circular-arc graphs. *SIAM J. Discrete Math.*, 7:647–655, 1994.
32. M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2:77–79, 1981.

A Modified cut-width for edge-weighted trees

As mentioned in the Introduction, PATHWIDTH can also be seen as a graph layout problem, and in this context pathwidth has been introduced under the name of vertex separation. There is a wide range of layout problems with various applications, see e.g. [14] for a survey. We are interested here in two of them, CUTWIDTH and MODIFIED CUTWIDTH.

Recall that *layout* $\mathcal{L}$ of a graph $G = (V, E)$ is a total ordering $(v_1, v_2, \ldots, v_n)$ of its vertices. Assume that G is an edge-weighted graph, i.e. we have a weight function associating to each edge xy a positive weight $w_e(xy)$. We denote by $\sigma_{\mathcal{L}}(x)$ the index of vertex x in the layout. The *cutwidth* of the layout $\mathcal{L}$ is

$$cwd(\mathcal{L}) = \max_{i, 1 \leq i \leq n} \sum_{xy \in E, \sigma_{\mathcal{L}}(x) \leq i < \sigma_{\mathcal{L}}(y)} w_e(xy)$$

In full words, the cutwidth of the layout is the maximum weight crossing some point i, where we say that an edge crosses i if one of the endpoints has an index at most i, and the other endpoint has an index strictly greater than i. The *cutwidth* of the edge-weighted graph G is the minimum cutwidth over all possible layouts of G:

$$\mathrm{cutwd}(G) = \min\{\mathrm{cutwd}(\mathcal{L}) \mid \mathcal{L} \text{ layout of } G\}$$

The *modified cutwidth* is defined in a very similar way except that this time, for an index i of a layout, we say that an edge crosses i if it starts strictly before i and ends strictly after i – in other terms edges incident to i are not counted. So the modified cutwidth of a layout $\mathcal{L}$ is

$$mcwd(\mathcal{L}) = \max_{i, 1 \leq i \leq n} \sum_{xy \in E, \sigma_{\mathcal{L}}(x) < i < \sigma_{\mathcal{L}}(y)} w_e(xy)$$

and the modified cutwidth of the edge-weighted graph G is

$$\mathrm{cutwd}(G) = \min\{\mathrm{cutwd}(\mathcal{L}) \mid \mathcal{L} \text{ layout of } G\}.$$

Both problems are NP-hard, even for unweighted graphs, in which case each edge counts 1 in the sums. Monien and Sudborough prove that CUTWIDTH is NP-hard for edge-weighted trees, even when weights are polynomial in the size of the tree. Based again on similar ideas, we show that MODIFIED CUTWIDTH is also NP-hard for weighted trees with polynomial weights. We point out that, despite the fact that that CUTWIDTH and MODIFIED CUTWIDTH are sintaxically very similar, there are not straightforward reductions between them. Somehow surprisingly, there are easy reductions between MODIFIED CUTWIDTH and PATHWIDTH, see e.g. [12], unfortunately these reductions do not stay in the class of trees. We prove:

Theorem 6. *The* MODIFIED CUTWIDTH *problem is NP-hard on edge-weighted trees, even when weights are polynomially bounded in the size of the tree.*

The proof is very similar to the one of Theorem 3, based on a reduction from BALANCED SYSTEM OF 0,1-LINEAR EQUATIONS:

Proposition 3. *Given a n instance* (A, b) *of* BALANCED SYSTEM OF 0,1-LINEAR EQUATIONS, *we construct the following edge-weighted tree* $T_e(A, b)$.

– *First construct an octopus tree with a head node h, also considered as the root of the tree, and n legs with m nodes each, where $m \times n$ is the size of matrix A. The nodes of leg j are denoted $n_{1,j}, n_{2,j}, \ldots, n_{m,j}$, starting from the head.*
– *Fix a number $k > 2b_1$ and, for each i, j, $1 \leq i \leq m$, $1 \leq j \leq n$ give the edge between $n_{i,j}$ and its parent the weight $A_{i,j}$.*
– *For each i, j, create three leaves $l^1_{i,j}, l^2_{i,j}, l^2_{i,j}$ adjacent to $n_{i,j}$. The weight of the three edges between $l^q_{i,j}$ and $n_{i,j}$ is $k + b_1 - b_i + A_{i,j} - A_{i+1,j}$ (here $A_{m+1,j} = 0$ for all j).*
– *create four leaves $l^1_h, l^2_h, l^3_h, l^4_h$ adjacent to the head node h. The corresponding incident edges are all of weight k.*

We have $\mathrm{modcutwd}(T_e(A, b)) \leq b_1 + k$ *if and only if the system $Ax = b$ has a 0,1-solution.*

Proof. Assume that the system $Ax = b$ has a $0, 1$-solution, we construct a layout $\mathcal{L}$ of $T_e(A, b)$ such that $\mathrm{modcutwd}(L) \leq k + b_1$. As usual, let x^* be a 0,1-solution of the system and let $L = \{j, 1 \leq j \leq n \mid x^*_j = 0\}$ and $R = \{j, 1 \leq j \leq n \mid x^*_j = 1\}$. Firstly we fix the ordering of the internal nodes of the tree. After the head node we put nodes corresponding to legs $j \in R$, in breadth-first search order starting from h: $n_{1,j_1}, n_{1,j_2}, \ldots, n_{1,j_p}$, $n_{2,j_1}, n_{2,j_2}, \ldots, n_{2,j_p}, \ldots,$ $n_{m,j_1}, n_{m,j_2}, \ldots, n_{m,j_p}$, where $R = \{j_1, j_2, \ldots, j_p\}$. The nodes of legs $\in L$ are put before h and ordered symmetrically w.r.t. h. We eventually insert the leaf nodes in the layout as follows. We insert two leaves incident to h right after h (and symmetrically two right before h). For each internal node $n_{i,j}$ with $j \in R$ we insert one of its pendant leaves, say $l^1_{i,j}$, right before the node, and the two other leaves $l^2_{i,j}$ and $l^3_{i,j}$ right after the node. (Thus for nodes $n_{i,j}$ with $j \in L$ we insert one of its leaves right after the node, and two of them right before the node).

It remains to show that $\mathrm{modcutwd}(\mathcal{L}) \leq k + b_1$. We say that an edge is *heavy* if its weight is at least k. Intuitively, "heavy edges" are much heavier than the "light" ones. Recall that an edge crosses a node u if one of its endpoints appears striclty before, and the other strictly after u in the layout. The weight crossing u is the sum of the weights of the edges crossing u. Let $n_{i,j}$ be a node situated after h in the layout. Among $n_{i,j}$ and the three adjacent leaves, clearly the position with maximum crossing weight is the leaf $l^2_{i,j}$ appearing right after $n_{i,j}$ (in particular the others are not crossed by heavy edges). The leaf $l^2_{i,j}$ is crossed by:

– a heavy edge incident to $n_{i,j}$, of weight $k + b_1 - b_i + A_{i,j} - A_{i+1,j}$;
– for each $j' \in R$ s.t. $n_{i,j'}$ appears strictly before $n_{i,j}$, the edge $n_{i,j'}n_{i+1,j'}$ of weight $A_{i+1,j'} < A_{i,j}$;
– for each $j'' \in R$ s.t. $n_{i,j'}$ appears strictly after $n_{i,j}$, the edge $n_{i-1,j''}n_{i,j''}$ of weight $A_{i,j''}$ (here $n_{0,j''} = h$);
– the edge $n_{i,j}n_{i+1,j}$, of weight $A_{i+1,j}$, if $i < m$.

Since $\sum_{j \in R} A_{i,j} = b_i$ it follows that the weight crossing $l^2_{i,j}$ is at most $k + b_1$. Among h and the incident leaves appearing after it in the layout, the maximum crossing weight corresponds to the leaf right after h. This weight is k (from the heavy edge) plus the sum $\sum_{j \in R} A_{1,j} = b_1$ (from the light edges incident to h). Again the crossing weight is at most $k + b_1$. By symmetry, the same holds for nodes appearing before h in the layout, hence $\mathrm{modcutwd}(\mathcal{L}) \leq k + b_1$.

Conversely, assume that $\mathrm{modcutwd}(T_e(A, b)) \leq k + b_1$.

Claim. There is an optimal layout $\mathcal{L}$ such that, for each internal node, all corresponding adjacent leaves appear next to the node in the layout. (Some of these leaves may apear before, and others after the internal node, but the node and its pendant leaves appear consecutively in the layout.)

Proof. (of the claim) First notice that, in a layout of modified cutwidth at most $k + b_1$, it is not possible that a heavy edge e crosses an internal node u. Indeed in this situation, there would be three heavy edges – e and two edges incident o u – such that the intersection of the intervals of the layout corresponding to the three edges is of length at least one. Only two of these three edges share an endpoint, thus in the layout we would have a position crossed by two heavy edges – a contradiction.

Let vl be a heavy edge, where v is the internal node. It is still possible that this edge crosses a leaf l' incident to some other node u. Then the only possibility is that v, l', l, u appear contiguously in this order, or the reverse one, otherwise two heavy edges cross a same position. Now we transform the layout replacing the sequence v, l', l, u by v, l, l', u without increasing the cutwidth. □

Claim. In the optimal layout $\mathcal{L}$ satisfying the previous claim no edge crosses the head node.

Proof. (of the claim) By the previous claim, we may assume that the four leaves incident to h are next to it in the optimal layout $\mathcal{L}$. Clearly, two of them should be to its left and the two others to its right. Let $L = \{j \mid n_{1,j} \text{ appears before } h\}$ and let $R = \{j \mid n_{1,j} \text{ appears after } h\}$ in the layout $\mathcal{L}$. Observe that all edges of type $n_{1,j}h$, with $j \in R$ (resp. $j \in L$) cross the leaf right after (resp. right before) h. This leaf is also crossed by a heavy edge of weight k. Thus $\sum_{j\in L} A_{1,j} \leq b_1$, also $\sum_{j\in L} A_{1,j} \leq b_1$, so both inequalities must by equalities. If some light edge crosses h, it also crosses the leaf right after and right before h, so the weight crossing these vertices exceeds $k + b_1$ – a contradiction. □

We can no define L (resp. R) the set of indices $j, 1 \leq j \leq n$, such that the nodes of type $n_{i,j}$ appear before (resp. after) h in the optimal layout. We prove that, for any i, $1 \leq i \leq m$, we have $\sum_{j\in R} A_{i,j} \leq b_i$, and by symmetry the same holds for L. For a fixed i, let $j_0 \in R$ be such that n_{i,j_0} is the leftmost element of the layout among all $n_{i,j}$, $j \in R$. Consider the leaf attached to $n_{i,j}$ which is crossed by a heavy edge incident to n_{i,j_0}; this leaf appears right before or right after n_{i,j_0}. The leaf is also crossed by edges of type $n_{i'-1,j}, n_{i',j}$ for all $j \in R \setminus j_0$ (again $n_{0,j} = h$) and some $i' \leq i$ depending on j. Each of these edges is of weight at least $A_{i,j}$. Finally, the same leaf crossed either by the edge n_{i-1,j_0}, n_{i,j_0}, of weight A_{i,j_0} (if this leaf is before n_{i,j_0}) or by edge n_{i,j_0}, n_{i+1,j_0}, of weight A_{i+1,j_0} (if the leaf is after n_{i,j_0} and $i < m$). Using the facts that $A_{i,j_0} > A_{i+1,j_0}$, the weight of the heavy edge is $k + b_1 - b_i + A_{i,j_0} - A_{i+1,0}$ and the total weight crossing the leaf should not exceed $k + b_1$ we deduce that $\sum_{j\in R} A_{i,j} \leq b_i$. Altogether, the system $Ax = b$ has a 0,1-solution x^* by taking $x^*_j = 0$ for all $j \in L$, and $x^*_j = 1$ for all $j \in R$. □

The tree $T_e(A, b)$ can be constructed in polynomial time w.r.t. the size of matrix A, which proves Theorem 6.