The Disjoint Paths Problem on Chordal Graphs

Frank Kammer and Torsten Tholey

Institut für Informatik, Universität Augsburg, D-86135 Augsburg, Germany
{kammer,tholey}@informatik.uni-augsburg.de

Abstract. Algorithms based on a bottom-up traversal of a tree decomposition are used in literature to develop very efficient algorithms for graphs of bounded treewidth. However, such algorithms can also be used to efficiently solve problems on chordal graphs, which in general do not have a bounded treewidth. By combining this approach with a sparsification technique we obtain the first linear-time algorithm for chordal graphs that solves the k-disjoint paths problem. In this problem k pairs of vertices are to be connected by pairwise vertex-disjoint paths. We also present the first polynomial-time algorithm for chordal graphs capable of finding disjoint paths solving the k-disjoint paths problem with minimal total length. Finally, we prove that the version of the disjoint paths problem, where k is part of the input, is $\mathcal{NP}$-hard on chordal graphs.

1 Introduction

In the *k-disjoint paths problem* (k-DPP), k pairs of vertices are to be connected by pairwise vertex-disjoint paths. This appears to be a hard problem since, for many classes of graphs, efficient algorithms are unknown or do not exist. Indeed, Fortune, Hopcroft, and Wyllie [3] have shown that the problem is $\mathcal{NP}$-hard on directed graphs, even if k is restricted to 2. As shown by Lynch [9] and by Knuth (see the paper of Karp [7]) the same is true on undirected graphs for the *disjoint paths problem* (DPP), where k, in contrary to the k-DPP, is part of the input. It is a common approach in combinatorial optimization to construct for $\mathcal{NP}$-hard problems so-called *fixed parameter algorithms* that solve the original problem in polynomial time if one or more of the input parameters are fixed. We present a linear time algorithm for the k-DPP, which then can be considered also as a fixed parameter algorithm for the DPP.

As usual in graph theory, we let n and m denote the number of vertices and edges, respectively, of the graph under consideration. For every fixed k, Robertson and Seymour, in their series of papers, developed a polynomial algorithm for the k-DPP on undirected graphs. Perković and Reed [13] presented an algorithm with an improved running time. Unfortunately, the constants hidden in the O-notation of the running time of the algorithms above are extremely large and make these algorithms unfeasible in practice. Algorithms with better practical running times are known for several classes of graphs such as undirected graphs of bounded treewidth [17] and directed acyclic graphs [3]. However, for many classes of graphs, e.g., for general, for planar, or for chordal undirected graphs,

algorithms more efficient than the algorithm of Perković and Reed are known only for the special case $k = 2$. The first polynomial-time algorithms for the case $k = 2$ on general undirected graphs are given, e.g., in [11, 18, 19, 21]. Perl and Shiloach [14] presented the first polynomial-time algorithm for the 2-DPP on undirected chordal graphs and on undirected planar graphs, namely with a linear running time. A simpler algorithm for chordal graphs can be found in [8].

The importance of chordal graphs is due to several facts. On the one hand, chordal graphs have nice properties that can be used to design efficient algorithms for many problems. For example, as shown by Fulkerson and Gross [4], chordal graphs are exactly the set of graphs with a perfect elimination order, and this order can be used to compute a maximal independent set, a maximum clique or an optimal coloring on chordal graphs in linear time [5]. On the other hand, concerning the practical relevance of chordal graphs, Gavril [6] has shown that the set of chordal graphs is equal to the set of *subtree graphs*, where a subtree graph is the intersection graph of a family of subtrees of a tree. Let us call a tuple $(G_1, \ldots, G_k, G)$ of graphs to be an intersection model for the intersection graph of $G_1, \ldots, G_k$ if the latter are subgraphs of G. Many practical problems in different areas such as computer science and genetics can be modeled by an intersection model and solved by a transformation to problems on the corresponding intersection graph; e.g., see [15]. In general, it seems that translating a problem on an intersection model into a problem on the corresponding intersection graph makes the problem easier to solve. However, in this paper we study the reverse direction. We translate the k-DPP on a chordal graph into a problem on the corresponding intersection model $(T_1, \ldots, T_k, T)$ or, more precisely, on a tree decomposition defined by this model, and we derive a simple approach to solve the k-DPP on chordal graphs. From another point of view our paper shows that algorithms based on a bottom-up traversal of a tree decomposition are useful not only for graphs of bounded treewidth, but can also be used for efficiently solving different problems on chordal graphs, even on those of unbounded treewidth. We only use the fact that we can choose the so-called bags of a tree decomposition as cliques. Following a similar approach, Okamato, Uno, and Uehara [12] have recently shown that the number of independent sets in a chordal graph can be counted in linear time.

In Section 2 we present an algorithm for solving the k-DPP on a chordal graph with a running time of $O(n^{2k+2} + m)$. As shown in Section 3, this algorithm can be modified to connect given pairs of vertices by pairwise disjoint paths such that the number of edges used by the paths is minimized among all such solutions. Note that so far no polynomial-time algorithm was known for solving this latter problem for every fixed k on chordal graphs.

In Section 4, as the main result of our paper, we show that the tree decomposition based algorithm of Section 2 can be combined with a sparsification technique in order to reduce the running time for solving the k-DPP on chordal graphs to $O(m + (2k)^{4k+2}n)$. This means that we obtain a linear fixed parameter algorithm for the DPP. Moreover, the additional constants hidden in the O-notation are of moderate size. For every fixed k, we obtain a running time

which improves the running time of the previous best known algorithm for solving the k-DPP on chordal graphs, namely the algorithm of Perković and Reed mentioned earlier in this introduction for solving the problem on general graphs. Moreover, our algorithm is easy to implement and—for small values of k—it is practical. It is not surprising that the running time increases exponentially in k since—as a further new result—we can proof that the DPP (with k being non-fixed) is $\mathcal{NP}$-hard even for chordal graphs. Details can be found in Section 5.

2 The k-Disjoint-Paths Problem

Many graph problems can be solved in polynomial time by traversing a so-called tree decomposition bottom-up if the so-called treewidth is taken to be a constant.

Definition 1. *A* tree decomposition *for a graph $G = (V, E)$ is a pair (T, B), where $T = (V_T, E_T)$ is a tree and B is a mapping that maps each node w of V_T to a subset $B(w)$ of V—called the* bag *of w—such that* **(1)** $\cup_{w \in V_T} B(w) = V$, **(2)** *for each edge $e \in E$, there exists a node $w \in V_T$ with $e \subseteq B(w)$,* **(3)** $B(x) \cap B(y) \subseteq B(w)$ *for all $w, x, y \in V_T$ with w being a node on the path from x to y in T. The* treewidth *of T is the maximal cardinality of a bag minus one and the* size *of a tree decomposition is the sum of the cardinalities of its bags.*

The *treewidth* of a graph G, denoted by $tw(G)$, is the smallest width of a tree decomposition for G. One of the problems that can be solved efficiently on graphs with constant treewidth is the k-disjoint paths problem [17]. Unfortunately, the treewidth of chordal graphs is not bounded by a constant but we can find a very special tree decomposition that helps us to solve the k-DPP even on chordal graphs. For a set $U \subseteq V$ of a graph $G = (V, E)$, we define $G[U]$ to be the subgraph of G induced by the vertices in U.

Definition 2. *A* clique tree for a graph $G = (V, E)$ is a tree decomposition (T, B) with the additional property that **(4)** B is a bijection between the nodes of T and $\{U \subseteq V \mid G[U]$ is a maximal clique in $G\}$.

It is well known that chordal graphs are exactly the graphs that have a clique tree [2, 6, 22] and that a clique tree of linear size can be constructed in linear time [1]. As one can show by using property (4) a clique tree has $O(|V|)$ nodes. As input of our algorithm we will take a *weak clique tree* that is defined as a clique tree if we replace (4) by the following property: **(4')** the vertices of each bag induce a clique in G. More precisely, our algorithm starts with constructing, for the graph $G = (V, E)$ on which we search for k disjoint paths, in $O(|V| + |E|)$ time a weak clique tree (T, B) of size $O(|V| + |E|)$ for G with T being a rooted tree having $O(|V|)$ nodes. For example we can take a standard clique tree. We call the vertices to be connected by disjoint paths the *terminals* of G. For a node w of T, we let $G(w)$ be the subgraph of G induced by all vertices contained in at least one set $B(w')$ for a descendant w' of w in T, where w is also a descendant of itself. In order to obtain a simpler description of our algorithm, we describe

our problem as a coloring problem. A *coloring* of a graph G' or a vertex set U is a mapping that assigns a color to some of the vertices in G' and in U, respectively. For an instance of the k-DPP, we always define an *initial coloring* that, for each pair of terminals to be connected, colors both terminals with the same color different from the colors of the other terminals. In general, we call a coloring C_2 of a vertex set V_2 an *extension* of a coloring C_1 of a vertex set $V_1 \subseteq V_2$ if all colored vertices of V_1 are also colored by C_2 with the same color. Let us call a coloring of a graph G *legal* if it is an extension of the initial coloring of the terminals and if, for each pair t_1 and t_2 of terminals sharing a color c, there is a path from t_1 to t_2 in the subgraph of G induced by the vertices of color c. Moreover, we call a legal coloring *good* with respect to a (weak) clique tree (T, B) for G if each bag of a node in T contains at most two vertices of color c.

For a node w with a father $p(w)$, let us call the set of vertices in $B(w) \cap B(p(w))$ the *transition set* of w denoted by $A(w)$. On the one hand, our algorithm will need to know the colors of all vertices in a bag for finally obtaining a complete coloring of G. On the other hand, our algorithm will extend stepwise colorings of $G(w)$ for a node w to colorings of $G(p(w))$ and for determining the colors for the new vertices of this extension (namely the vertices in $G(p(w)) - G(w)$) it only needs to know the colors of the vertices in $A(w)$. Thus, we define a *full* and a *reduced characteristic* for a node w of T as a coloring of the nodes in $B(w)$ and $A(w)$, respectively, such that for each color c at most two vertices are colored with c. Let Z be a full or a reduced characteristic of a node w. Then Z is *valid* if and only if

1. there exists a good coloring C of $G(w)$ extending Z and the initial coloring of the terminals in $G(w)$.
2. for each color c the following is true: if there is exactly one terminal in $G(w)$ of color c, a vertex in $A(w)$ is colored with c by Z.

A coloring C with properties 1 and 2 is then called *conform* to Z. We also call two characteristics *compatible* if one characteristic is an extension of the other.

There is a connection between the k-DPP and good colorings: If the k-DPP has a solution, take a solution with minimal total length. Then a coloring that colors the vertices of each path of the solution with the color of its endpoints is a good coloring because the following is true: if one of the disjoint paths visits three vertices v_1, v_2, and v_3 of one bag in this order, we obtain a shorter path by replacing the subpath from v_1 to v_3 by edge $\{v_1, v_3\}$. In the reverse direction assume that we are given a good coloring conform to a valid full characteristic of the root of T. Then disjoint paths connecting the terminals of the same color can be obtained by a depth-first search on each subgraph induced by the vertices of one color. Hence, we can solve the k-DPP by computing a good coloring conform to a valid full characteristic of the root of T.

For all nodes of T, we want to determine bottom-up all valid full and all valid reduced characteristics. If we restrict a coloring of $G(w)$ conform to a valid full characteristic of a node w to the graph $G(w')$ for a descendant w' of w, this restricted coloring is conform to a valid full as well as to a valid reduced

characteristic of w'. In the reverse direction, a full characteristic Z of a node w is valid if and only if

- the terminals in $B(w)$ are colored by Z with their original color,
- for each child w' of w, the reduced characteristic of w' compatible to Z is valid,
- each color is used by Z to color at most two vertices, and
- for each color c, the following is true: if there is exactly one terminal in $G(w)$ of color c, a vertex in $A(w)$ is colored with c by Z.

Because of the above conditions there exists a good coloring of $G(w)$ extending Z; in particular, using the last condition we can conclude by induction that, for each pair of terminals t_1 and t_2 sharing a color c, there is a path from t_1 to t_2 in the subgraph of G induced by the vertices of color c. By iterating over all valid full characteristics of a node w in T we can easily compute a lookup-table storing 1 for each valid reduced characteristic of w and 0 for each non-valid reduced characteristic of w. The time for updating the whole table for w is of size $O(\ell \cdot tw(G)\deg(w))$, where ℓ is the number of full characteristics of w.

Hence, by a bottom-up traversal of T we can find a good coloring for G if such a coloring exists. We next analyze the running time of our algorithm. For each node, we can test whether a certain full characteristic is valid in at most $O((tw(G)+k)\deg(w))$ time by testing the four properties listed above. For testing the last condition, note the following: in $O(k\deg(w))$ time, we can update the set of colors c for which there is exactly one terminal of color c in $G(w)$ if we are given the corresponding sets for the children of w. There are at most $(tw(G)+1)^{2k}$ full characteristics for a node w since for each color we can choose twice either no vertex or one of the $\leq tw(G)+1$ vertices in $B(w)$. Since the time needed to initialize the lookup-table for the reduced characteristics of a node w is bounded linear in $O(|A(w)|)$ times the number of full characteristics, we obtain the next lemma.

Lemma 3. *The k-DPP can be solved in* $O((tw(G)+1)^{2k}(tw(G)+k)|V|+|E|) = O(|V|^{2k+2}+|E|)$ *time on a chordal graph* $G=(V,E)$.

3 Shortest *k*-Disjoint Paths

Define the weight of a coloring as the number of edges whose endpoints are both colored and have the same color. The cost of a characteristic Z of a node w is the minimal possible weight of a coloring of $G(w)$ conform to Z. In order to output disjoint paths using a minimal number of edges, we also have to compute the costs of the characteristics. Given, for a full characteristic Z of a node w, the costs $W(Z_1),\ldots,W(Z_\ell)$ of the reduced characteristics $Z_1,\ldots,Z_\ell$ of the children of w compatible to Z, one can compute the cost $W(Z)$ of Z as follows: Initialize $W(Z)$ with $W(Z_1)+W(Z_2)+\ldots+W(Z_\ell)$. Subsequently, for each color c used by Z to color two vertices $v_1, v_2 \in B(w)$, add one minus the number of children of w with their bags containing both, v_1 and v_2. This update takes $O(\min\{k,|B(w)|\}\cdot \deg(w))$ time and does not increase the asymptotic running time.

Theorem 4. *On a chordal graph $G = (V, E)$ one can find in $O(|V|^{2k+2} + |E|)$ time paths solving an instance of the k-DPP using a minimal number of edges among all sets of paths solving the instance.*

4 A Speedup

In this section we present a speed-up of the algorithm in Section 2. Once again, we first construct in $O(|V|+|E|)$ time a weak clique tree (T, B) of size $O(|V|+|E|)$ for our input graph $G = (V, E)$ with T having $O(|V|)$ nodes. We assume that there is no edge $\{t_1, t_2\}$ in G for a pair $\{t_1, t_2\}$ of terminals that are to be connected in G. Otherwise, our problem would be reduced to the $(k-1)$-DPP on $G[V - \{t_1, t_2\}]$. For each pair (t_1, t_2) of terminals t_1 and t_2 that should be connected by a path, let us choose $\Gamma(t_1)$ and $\Gamma(t_2)$ as the unique pair of nodes in T with their bags containing t_1 and t_2, respectively, such that the distance between the nodes is minimal. We choose for an arbitrary terminal t the node $\Gamma(t)$ as the root of T. Let f be a fixed bijection from V to $\{1, \ldots, |V|\}$ assigning the highest numbers to the terminals of G. For nodes w_1 and w_2 of T and for a vertex v of G, we define the $(w_1, w_2)_B$-*count* of v as a tuple $(a, f(v))$, where a is the number of nodes w' on the path from w_1 to w_2 in T whose bags $B(w')$ contain v. We say that a vertex v_1 with $(w_1, w_2)_B$-count (a_1, b_1) has a larger $(w_1, w_2)_B$-count than a vertex v_2 with $(w_1, w_2)_B$-count (a_2, b_2) if and only if either $a_1 = a_2$ and $b_1 > b_2$ or $a_1 > a_2$ holds. For a node $w \in T$, we let $I(w) = \{t \mid t \text{ is terminal with } \Gamma(t) \text{ contained in the subtree of } T \text{ rooted in } w\}$.

In order to improve the efficiency of the algorithm presented in Section 2, we replace (T, B) by a new tuple (T^*, B^*), where T^* will be a subtree of T and where B^* will be a function that maps each node w of T^* to a subset of $B(w)$ of size $\leq 4k^2$. In order to describe (T^*, B^*) more precisely, we need some further definitions. A bag $B(w)$ of a node w is called *small* if $|B(w)| \leq 2k$ and *big* otherwise. For each node w of T and for each terminal t, we define $D(w, t)$ as the set of the $\min\{2k, |B(w)|\}$ vertices of $B(w)$ with the largest $(w, \Gamma(t))_B$-count. We also let $D(w)$ be the union of $D(w, t)$ over all $t \in I(w)$ and of the set of all terminals in $B(w) \setminus I(w)$.

We now obtain T^* from T by deleting all nodes w with $I(w) = \emptyset$. We choose the same root for T^* as for T and, for each node w, we insert the vertices of $D(w)$ into $B^*(w)$. Moreover, for each child w' of w, we insert an arbitrary subset of $D(w) \cap B(w')$ of size $\min\{2k, |D(w) \cap B(w')|\}$ into $B^*(w')$. Let $t \in I(w')$. Keep in mind that, if $|B(w) \cap B(w')| \geq 2k$, then $D(w, t)$—and consequently also $B^*(w)$—contains $2k$ vertices of $B(w')$ since these vertices have the largest $(w, \Gamma(t))_B$-count. Thus, if $|B(w) \cap B(w')| \geq 2k$, the rules for node w add $2k$ vertices of $B^*(w)$ to $B^*(w')$, i.e., $|B^*(w) \cap B^*(w')| \geq 2k$. Note that by our definition $B(w) = B^*(w)$ holds for each small bag $B(w)$. We also can conclude:

Lemma 5. *Let v be a vertex of G, w' be a node of T^* with $v \in B^*(w')$, and w'' be the node of lowest depth with $v \in B(w'')$. Then $v \in B^*(w)$ holds for each node w on the path from w' to w'' in T^*.*

Proof. Since $B(w)$ and $B^*(w)$ share the same terminals, Lemma 5 holds if v is a terminal. If it is not, we merely need to show that, for each node w on the path from w' to w'' in T with $v \in B(w)$, the following holds: if w has a father x with $v \in B(x)$, we also have $v \in B^*(x)$. Since $B^*(x) = B(x)$ if $B(x)$ is small, we only need to consider the case, where $B(x)$ is big. Let us consider the case, where $B(w)$ is small. Because of $v \in B^*(w)$ there is a $t \in I(w)$ for which $v \in D(w,t)$ or $v \in D(x,t)$ holds. Since $|B(w)| \leq 2k$, $v \in D(w,t) \cap B(x)$ also implies $v \in D(x,t)$. Consequently, $v \in B^*(x)$. Let us finally consider the case where both, $B(w)$ and $B(x)$, are big. If the insertion rule for x inserts v into $B^*(w)$, we have $v \in B^*(x)$. Otherwise, the only reason for v being contained in $B^*(w)$ is that $v \in D(w,t)$ for a terminal $t \in I(w)$. Then v must also be one of the vertices with the $2k$ largest $(x, \Gamma(t))_B$-counts and therefore is also contained in $B^*(x)$. □

Corollary 6. *For each vertex v of G, the subtree of T^* induced by the nodes w with $v \in B^*(w)$ is connected.*

Let G^* be the graph obtained from G by removing all vertices v and all edges $\{v_1, v_2\}$ from G for which there is no longer any node w with $v \in B^*(w)$ and $\{v_1, v_2\} \subseteq B^*(w)$, respectively.

Lemma 7. *(T^*, B^*) is a weak clique tree for G^* of width $4k^2 - 1$.*

Proof. By our construction and Corollary 6 all properties of a weak clique tree hold for (T^*, B^*). Concerning the treewidth, for the root r of T, we have $|B^*(r)| \leq |D(r)| \leq 4k^2$ since $|I(r)| = 2k$. By our choice of r there is a terminal t_1 with $\Gamma(t_1) = r$. We have $|D(w)| \leq 4k^2 - 2k$ for all nodes $w \neq r$ in T since the subtree of T rooted in w does not contain $\Gamma(t_1)$. Consequently, $|B^*(w)| \leq 4k^2$. □

Lemma 8. *The k-DPP has a solution on G if and only if this is true for G^*.*

Proof. Clearly, an instance of the k-DPP is solvable on G if this true for G^*. For the reverse direction we merely need to show that a solution of the k-DPP on G allows us to construct a good coloring of G^* with respect to (T^*, B^*). Moreover, for a legal coloring C and a pair of terminals t_1 and t_2 colored with c by C, let us call a pair of incident nodes w_1 and w_2 on the unique path from $\Gamma(t_1)$ to $\Gamma(t_2)$ a *color break* with respect to c (and C) if no vertex in $B^*(w_1) \cap B^*(w_2)$ is colored with c. Let $\mathcal{C}$ be the set of all legal colorings of G that color at most two vertices of each bag in (T^*, B^*) with the same color. The solvability of the k-DPP implies $\mathcal{C} \neq \emptyset$ since there exists—as shown in Section 2—at least one good coloring with respect to (T, B) and since each good coloring is contained in $\mathcal{C}$. In the reverse direction, our Lemma holds if there is a $C \in \mathcal{C}$ without any color break since C then is a good coloring of G^* with respect to (T^*, B^*).

Assume now that we can find no coloring in $\mathcal{C}$ without color breaks. Let us choose a fixed numbering with $1, \ldots, k$ for the colors assigned to the terminals and a coloring $C \in \mathcal{C}$ such that the lowest number among the colors with a color break is as large as possible. Moreover, if c is the color with the lowest number for which there is a color break and if t_1 and t_2 are the terminals of color c, we

choose the coloring $C \in \mathcal{C}$ with the above properties such that there is a maximal distance between $\Gamma(t_1)$ and the node w_{σ_0} of the first color break $(w_{\sigma_0}, w_{\sigma_0+1})$ on the path $p = (w_1, w_2, w_3, \ldots)$ from $\Gamma(t_1)$ to $\Gamma(t_2)$ in T. Let v be a vertex of color c with $v \in B(w_{\sigma_0}) \cap B(w_{\sigma_0+1})$ and $v \notin B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$.

Assume $|B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})| < 2k$. Then $|B(w_{\sigma_0}) \cap B(w_{\sigma_0+1})| < 2k$. Let $w \in \{w_{\sigma_0}, w_{\sigma_0+1}\}$ be the father of the other node $w' \in \{w_{\sigma_0}, w_{\sigma_0+1}\}$ and $t \in \{t_1, t_2\} \cap I(w')$. We can conclude $v \in D(w, t)$ and consequently $v \in B^*(w)$. Since $|B(w_{\sigma_0}) \cap B(w_{\sigma_0+1})| < 2k$, the rule for w adds all vertices of $D(w, t) \cap B(w')$ including v into $B^*(w')$, a contradiction to $v \notin B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$.

Hence $|B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})| \geq 2k$. Since no vertex in $B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$ is colored with c and since C is a coloring which uses each color at most twice in a bag of (T^*, B^*) and thus in $B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$, it follows that there must be an uncolored vertex in $B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$. Let us define u to be the uncolored vertex in $B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$ that among all uncolored vertices has the largest $(w_{\sigma_0+1}, \Gamma(t_2))_B$-count. We next show that we can construct a coloring $C^* \in \mathcal{C}$ without any new color breaks for the colors different from c for which—if it has a color break with respect to c—the first such color break occurs after the pair $\{w_{\sigma_0}, w_{\sigma_0+1}\}$ on p. This leads to a contradiction to our choice of C and proofs our lemma.

After having initially set $C^* = C$ we modify C^* as follows. First of all, we color u with c. We then define $w_{\sigma_{-1}}$ and w_{σ_1} as the first and the last node on p, respectively, such that $u \in B^*(w_{\sigma_{-1}}) \cap B^*(w_{\sigma_1})$. Let S be the set consisting of u and all vertices colored with c by C contained in a bag of $\{B(w_{\sigma_{-1}}), \ldots, B(w_{\sigma_1})\}$.

Second, modify C^* such that it does not color the vertices in S apart from u, the vertex $v' \in S$ with the largest $(w_{\sigma_{-1}}, \Gamma(t_1))_{B^*}$-count, and the vertex $v'' \in S$ with the largest $(w_{\sigma_1}, \Gamma(t_2))_B$-count, where possibly $|\{u, v', v''\}| < 3$. Note that C being a legal coloring implies $v'' \in B(w_{\sigma_1+1})$ or $w_{\sigma_1} = \Gamma(t_2)$.

Third, if $v'' \in B(w_{\sigma_0+1}) \setminus D(w_{\sigma_0+1}, t_2)$, no vertex is colored with c by C in $D(w_{\sigma_0+1}, t_2)$. Then, let $\tilde{u}$ be the vertex of highest $(w_{\sigma_0+1}, \Gamma(t_2))_B$-count in $D(w_{\sigma_0+1}, t_2)$ not colored by C, let w_{σ_2} be the last node on p with $\tilde{u} \in B(w_{\sigma_2})$, and let v''' be the vertex in $B(w_{\sigma_2})$ with $C(v''') = c$ that among all such vertices has the largest $(w_{\sigma_2}, \Gamma(t_2))_B$-count. In particular, $v''' \in B(w_{\sigma_2+1})$ or we have $w_{\sigma_2} = \Gamma(t_2)$ and $v''' = t_2$. If $v'' \in B(w_{\sigma_0+1}) \setminus D(w_{\sigma_0+1}, t_2)$ and $v'' \neq u \neq \tilde{u}$, so-called *extra modifications* of C^* are required: change C^* such that it does not color v'' nor any other vertices colored with c by C and being contained in a bag of $\{B(w_{\sigma_1}), \ldots, B(w_{\sigma_2})\}$ except u, $\tilde{u}$, and v'''. See Fig. 1. (Some ranges are explained later in more detail.) Coloring v' with c guarantees that there is no new color break between $\Gamma(t_1)$ and w_{σ_0} on p.

Let us first consider the case, where no extra modifications are applied. By coloring v'' with c we can guarantee that C^* is a legal coloring. Hence, if C^* does not belong to $\mathcal{C}$, there is a bag $B^*(w)$ containing u, v', v'' and $|\{u, v', v''\}| = 3$. By Corollary 6 we can choose w w.l.o.g. as a node on p. Due to our choice of v' we know that $v' \in B^*(w_{\sigma_{-1}})$. There is no node w' on the subpath of p from w_{σ_0+1} to $\Gamma(t_2)$ with $v' \in B^*(w')$ since otherwise $v' \in B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$

by Corollary 6. Thus, w is on the subpath of p from $\Gamma(t_1)$ to w_{σ_0}. Note that $v'' \in B^*(w)$ implies $v'' \in B(w)$. We consider two subcases:

- w_{σ_0+1} is the father of w_{σ_0}. Since $v'' \in B(w_{\sigma_1})$, we also have $v'' \in B(w_{\sigma_0}) \cap B(w_{\sigma_0+1})$. Therefore, $v'' \in B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$ according to Lemma 5 and we obtain a contradiction since $(w_{\sigma_0}, w_{\sigma_0+1})$ is a color break.
- w_{σ_0} is the father of w_{σ_0+1}. Since no extra modifications are applied and since $u \neq v''$, we have $v'' \notin B(w_{\sigma_0+1})$ or $v'' \in D(w_{\sigma_0+1}, t_2) \subseteq B^*(w_{\sigma_0+1})$ or $u = \tilde{u}$. In the first case, v'' has a smaller $(w_{\sigma_1}, \Gamma(t_2))_B$-count than u. In the second case, we have $v'' \in B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$ by Corollary 6. If only the third case holds, because of $u \in D(w_{\sigma_0+1}, t_2)$ and $v'' \notin D(w_{\sigma_0+1}, t_2)$, vertex u has a larger $(w_{\sigma_1}, \Gamma(t_2))_B$-count than v''. Consequently, a contradiction occurs in each case.

Let us finally assume that the extra modifications are being applied. Then $u \neq \tilde{u}$ and hence $\tilde{u} \notin B^*(w_{\sigma_0})$. Like in the previous case without extra modifications, v' can not be contained in one of the bags $B^*(w_{\sigma_0+1}), \ldots, B^*(\Gamma(t_2))$. Thus, no bag of (T^*, B^*) contains v' and $\tilde{u}$. Since $D(w_{\sigma_0+1}, t_2) \cap \{v'', \tilde{u}\} = \{\tilde{u}\}$, the $(w_{\sigma_0+1}, \Gamma(t_2))_B$-count of v'' is smaller than that of $\tilde{u}$. Therefore and because of $v'' \in B(w_{\sigma_0+1})$, the $(w_{\sigma_1}, \Gamma(t_2))_B$-count of v'' is smaller than that of $\tilde{u}$. Note that $v''' \notin B(w_{\sigma_1})$ since otherwise $v''' \in S$ and the fact that $v''' \in B(w_{\sigma_2+1})$ or $v''' = t_2$ would imply that v''' has a larger $(w_{\sigma_1}, \Gamma(t_2))_B$-count than that of $\tilde{u}$ and that of v''. Thus, no bag of (T, B) contains u and v'''. Consequently, C^* colors at most two vertices in each bag of (T^*, B^*). Our choice of v''' guarantees that C^* is a legal coloring.

We have shown that $C^* \in \mathcal{C}$ and that the distance between $\Gamma(t_1)$ and the first node of a color break on p with respect to c and C^*—if indeed there is a color break—is larger than the corresponding distance for C. This is a contradiction to our choice of C. □

We can therefore solve the k-DPP on a chordal graph G as follows: we first determine for each node w of T the set $I(w)$ and subsequently all sets $D(w, s)$ and $D(w)$ ($s \in I(w)$). This can easily be done in a bottom-up traversal of T in at

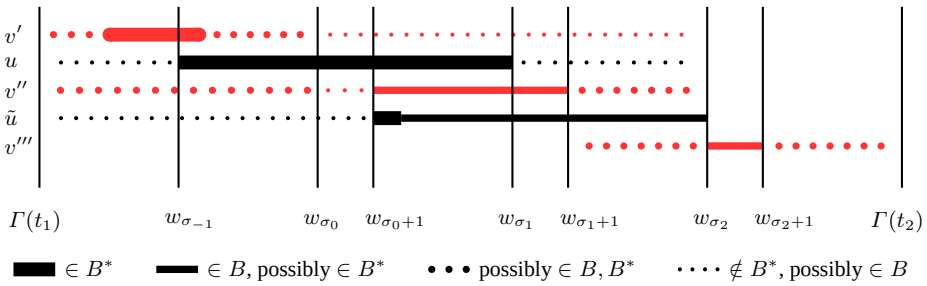

Fig. 1. The ranges of the variables in the case where the extra modifications are applied. Black lines represent vertices uncolored by C.

most $O(m+k^3n)$ time. Hence, we can replace G and (T, B) by G^* and (T^*, B^*) in the same running time. We then apply the algorithm of the Section 2 on G^*.

Theorem 9. *The k-DPP on chordal graphs with n vertices and m edges can be solved in $O(m + (4k^2)^{2k+1}n) = O(m + (2k)^{4k+2}n)$ time.*

5 Hardness of the Disjoint-Paths Problem

Theorem 10. *The disjoint-paths problem on chordal graphs is NP-hard.*

Proof. We can prove the theorem by a reduction from a restricted case of *1-in-3 SAT*. In 1-in-3 SAT we are given a formula in conjunctive normal form with 3 variables per clause and we have to find an assignment of the variables such that exactly one of the three literals is true in every clause. A formula in conjunctive normal form is *monotone* if every literal is positive and it is *cubic* if every variable occurs exactly three times. In [10] it is shown that 1-in-3 SAT is NP-complete even on monotone and cubic formulas. We now reduce an instance of 1-in-3 SAT consisting of a monotone and cubic formula F to an instance of the DPP on a chordal graph G. Fig. 2 should represent a clique tree of G. Each subgraph induced by the vertices of a bag should be a clique whose edges are colored gray in Fig. 2—however, not all existing edges are shown in the figure. Black lines represent paths of length 0. Therefore, the endpoints of black lines represent the same vertex even if they appear in different shapes.

In detail, we construct G as follows: For each variable x and each clause C in F, we introduce a *variable* and a *clause gadget*, respectively, as shown in Fig. 2. A variable gadget has six terminals $a_1, a_2, a_3, b_1, b_2, b_3$ and a clause gadget six terminals $y_1, y_2, y_3, z_1, z_2, z_3$. Each gadget is connected to one big clique Γ—see the rightmost bag Fig. 2. Γ contains 6ℓ vertices where ℓ is the number of clauses. We next divide the terminals into pairs such that the resulting instance of the DPP has a solution if and only if F has a satisfying assignment. If a clause C contains a variable x as the i-th variable and if it is the j-th occurrence of variable x in F that is part of C, the pairs (a_j, y_i) and (b_j, z_i) are added to our instance of the DPP, where the four terminals a_j, b_j, y_i, and z_i belong to the gadgets for x and C. Moreover, we identify one triangular and one square vertex in the gadget of x with one triangular and one square vertex, respectively, in the gadget of C different from the triangular and square vertices chosen for other variables or clauses. For a simpler notation, the terminals a_1, a_2 and a_3 shown in Fig. 2 are called A-terminals and the remaining terminals B-, Y- and Z-terminals, respectively.

Let us consider a satisfying assignment of F. For a variable x, we construct in the gadget of x six paths from terminals to the triangular and square vertices such that, if x is set to true, the paths starting in the A-terminals are routed exclusively to the triangular vertices; otherwise, they are routed exclusively to the square vertices. Since each clause C has exactly one true variable, for each clause gadget, exactly one path from an A-terminal to the gadget of C arrives at a triangular vertex, whereas the other two paths from an A-terminal to the

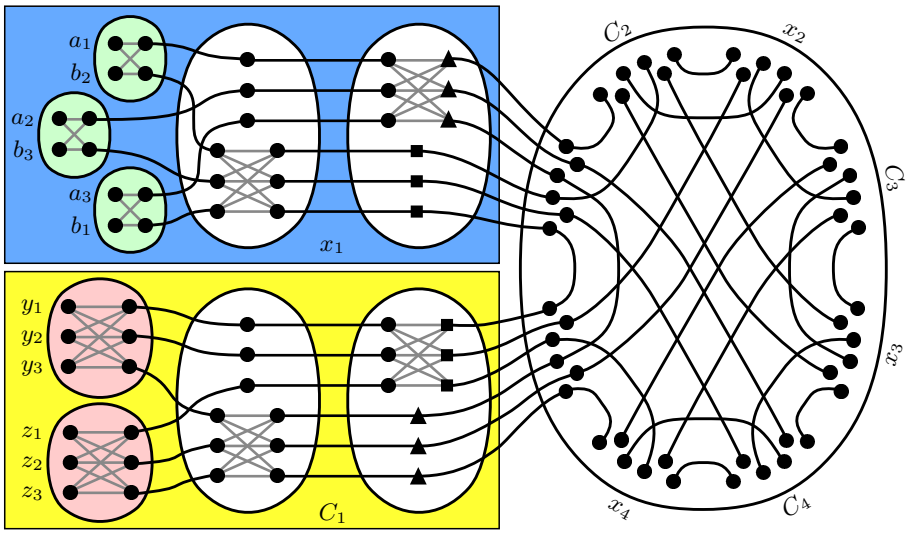

Fig. 2. Reduction from $(x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4)$.

gadget of C arrive at a square vertex. Thus, we can forward the three paths to the Y-terminals of C. Similarly, we can forward the paths from the B-terminals to the Z-terminals. Hence, we have found a solution to our instance of the DPP.

Let us now consider a solution of our DPP. It remains to show that F can be satisfied. In our construction, the number of vertices in the big clique Γ is equal to the number of pairs that have to be connected in our instance. Moreover, each path has to use at least one and therefore exactly one vertex of Γ. Note that for each clause C and each variable x of C, Γ contains exactly two vertices common with the gadgets of C and x, respectively: one triangular and one square vertex. Thus, these two vertices must be the two vertices visited by the two paths connecting two pairs of terminals in the gadgets of C and x. As a consequence, for any fixed variable gadget the paths starting from the A-terminals must pass either exclusively through the triangular vertices or exclusively through the square vertices of this gadget—see the variable gadget in Fig. 2. We define a variable x of F to be true if the paths of the A-terminals from the gadget of x pass through the triangular vertices. Since the A-terminals are connected to the Y-terminals, exactly one path from an A-terminal uses a triangular vertex of each clause gadget, i.e., exactly one variable of each clause is set to true. □

References

1. J. R. S. Blair, R. E. England, and M. G. Thomason, Cliques and their separators in triangulated graphs, Tech. Report UT-CS-88-73, Dept. Comput. Sci., University of Tennessee, Knoxville, 1988.

2. P. Buneman, A characterisation of rigid circuit graphs, *Discrete Math.* **9** (1974), pp. 205–212.
3. S. Fortune, J. Hopcroft, and J. Wyllie, The directed subgraph homeomorphism problem, *Theoret. Comput. Sci.* **10** (1980), pp. 111–121.
4. D. R. Fulkerson and O. A. Gross, Incidence matrices and interval graphs, *Pacific J. Math.* **15** (1965), pp. 835–855.
5. F. Gavril, Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph, *SIAM J. Comput.* **1** (1972), pp. 180–187.
6. F. Gavril, The intersection graphs of subtrees in trees are exactly the chordal graphs, *J. Combin. Theory Ser. B* **16** (1974), pp. 47–56.
7. R. M. Karp, On the computational complexity of combinatorial problems, *Networks* **5** (1975), pp. 45–68.
8. S. V. Krishnan, C. Pandu Rangan, and S. Seshadri, A new linear algorithm for the two path problem on chordal graphs, Proc. 8th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1988), LNCS Vol. 338, Springer, Berlin, 1988, pp. 49–66.
9. J. F. Lynch, The equivalence of theorem proving and the interconnection problem, *(ACM) SIGDA Newsletter* **5** (1975), pp. 31–36.
10. C. Moore and J. M. Robson, Hard tiling problems with simple tiles. *Discrete and Comput. Geom.* **26** (2001), pp. 573-590.
11. T. Ohtsuki, The two disjoint path problem and wire routing design, Proc. Symposium on Graph Theory and Algorithms, LNCS, Vol. 108, Springer, Berlin, 1981, pp. 207–216.
12. Y. Okamoto, T. Uno, and R. Uehara, Linear-time counting algorithms for independent sets in chordal graphs, Proc. 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2005), LNCS Vol. 3787, Springer, Berlin, 2005, pp. 433–444.
13. L. Perković and B. Reed, An improved algorithm for finding tree decompositions of small width, *International Journal of Foundations of Computer Science (IJFCS)* **11** (2000), pp. 365–372.
14. Y. Perl and Y. Shiloach, Finding two disjoint paths between two pairs of vertices in a graph, *J. ACM* **25** (1978), pp. 1–9.
15. F. S. Roberts, Discrete Mathematical Models with Applications to Social, Biological, and Environmental Problems. Prentice Hall, Englewood Cliffs, NJ, 1976.
16. N. Robertson and P. D. Seymour, Graph minors. XIII. The disjoint paths problem, *J. Comb. Theory Ser. B* **63** (1995), pp. 65–110.
17. P. Scheffler, A practical linear time algorithm for disjoint paths in graphs with bounded tree-width, Report No. 396/1994, TU Berlin, FB Mathematik, 1994.
18. P. D. Seymour, Disjoint paths in graphs, *Discrete Math.* **29** (1980), pp. 293–309.
19. Y. Shiloach, A polynomial solution to the undirected two paths problem, *J. ACM* **27** (1980), pp. 445–456.
20. R. E. Tarjan and M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, *SIAM J. Comput.* **13** (1984), pp. 566–579.
21. C. Thomassen, 2-linked graphs, *European J. Combin.* **1** (1980), pp. 371–378.
22. J. R. Walter, Representations of chordal graphs as subtrees of a tree, J. Graph Theory **2** (1978), pp. 265–267.