Two-Layer Planarization Parameterized by Feedback Edge Set*

Johannes Uhlmann and Mathias Weller

Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{johannes.uhlmann,mathias.weller}@uni-jena.de

Abstract. Given an undirected graph G and an integer $k \geq 0$, the NP-hard 2-Layer Planarization problem asks whether G can be transformed into a forest of caterpillar trees by removing at most k edges. Since transforming G into a forest of caterpillar trees requires breaking every cycle, the size f of a minimum feedback edge set is a natural parameter with $f \leq k$. We improve on previous fixed-parameter tractability results with respect to k by presenting a problem kernel with $O(f)$ vertices and edges and a new search-tree based algorithm, both with about the same worst-case bounds for f as the previous results for k, although we expect f to be smaller than k for a wide range of input instances.

1 Introduction

The focus of this work is on finding good 2-layered drawings of graphs. Such drawings appear in the "Sugiyama" approach [17,13] to multilayered graph drawing [5,8].

In this context, a graph is called *biplanar* if it can be drawn in two layers without edge crossings (while edges are drawn as straight lines). It has been shown that biplanar graphs are exactly the graphs that consist of disjoint caterpillar trees (or caterpillars for short) [13,5]. A caterpillar is a tree where every vertex is adjacent to at most two non-leaf vertices. In this work, we concentrate on the NP-hard 2-Layer Planarization (2LP) problem, that is, the problem to transform a given graph into a forest of caterpillars by deleting a minimum number of edges. Formally, this problem is defined as follows. Given an undirected graph $G = (V, E)$ and a non-negative integer k, determine whether there is an edge subset $E' \subseteq E$ with $|E'| \leq k$ such that $(V, E \setminus E')$ is biplanar.

Apart from being proposed as an alternative method to minimize crossings [13], solving 2LP is important in DNA mapping [19] and global routing for row-based VLSI layout [12]. 2LP is NP-hard even in the case that the input graph is bipartite and in one partition each vertex has degree at most two [7]. Shahrokhi et al. [15] presented a dynamic programming based linear-time algorithm solving the problem on trees. Concerning the parameter k (number of edge

* Supported by the DFG, research projects PABI (NI 369/7) and DARE (GU 1023/1, NI 369/11).

deletions), Dujmović et al. [5] showed that 2LP can be solved in $O(k \cdot 6^k + |G|)$ time by devising a search tree algorithm and several polynomial-time data reduction rules leading to a problem kernel with $O(k)$ vertices and edges. Later, Fernau [8] presented a refined search tree for 2LP leading to a running time of $O(k^2 \cdot 5.19276^k + |G|)$. Finally, based on a different branching analysis, Suderman [16] developed an $O(k \cdot 3.562^k + |G|)$-time algorithm.

2LP is a special case of the problem of transforming a binary matrix into a matrix with so-called "consecutive ones property" by a minimum number of column removals. More specifically, 2LP coincides with this problem for matrices without identical columns that have a maximum of two 1s in each column [3].

In this work, we investigate the parameterized complexity of 2LP with respect to the parameter "feedback edge set number" f, that is, the minimum number of edges whose removal results in an acyclic graph. Note that the feedback edge set number of a connected n-vertex and m-edge graph is $f(G) = m - n + 1$ and a minimum feedback edge set can be determined by the computation of a spanning tree in $O(n + m)$ time via depth-first search. We develop efficient preprocessing rules for 2LP that lead to a problem kernel with $O(f)$ vertices and $O(f)$ edges. Moreover, we present a new search tree algorithm leading to a total running time of $O(6^f + f \cdot m)$ for solving 2LP.

Our work is motivated as follows. First, note that for 2LP the number of necessary edge deletions is at least the feedback edge set number, since one has to destroy all cycles to obtain a forest of caterpillars. In this sense, we improve on the results of Dujmović et al. [5] by providing fixed-parameter algorithms and kernelizations with about the same worst-case bounds for a parameter that we expect to be significantly smaller for a wide range of input instances. Second, Dujmović et al. [5] pointed out that "instances of 2-Layer Planarization for *dense* graphs are of little interest from a practical point of view" since the resulting drawings are unreadable anyway. Thus, they expect the solution size to be small in practice. This is even more plausible for the feedback edge set number of a graph which is directly linked to the number of edges, and, hence, the sparseness of the graph. Also note that the solution size can be arbitrarily large even for trees (the sparsest connected graphs) while the feedback edge set number of trees is zero. Measuring the distance from trees by the feedback edge set number can be seen as a parameterization by "distance from triviality" [10]. In this sense, our results generalize the liner-time algorithm for trees by Shahrokhi et al. [15]. Third, the feedback edge set number f is a parameter that can easily be computed in advance and, hence, allows for a meta-algorithm that chooses an algorithm for a given input by computing an estimation on the running time prior to running the algorithm for the problem itself. Since the parameter k ("number of edge deletions") is NP-hard to compute, such an algorithm could not efficiently determine the running time of an algorithm parameterized by k in advance. Fourth, looking for smaller parameters may help to further extend the range of solvable instances. However, this seems only possible if the new algorithms have a modest exponential part of the running time.

Last but not least, efficient preprocessing or polynomial-time data reduction seems to be essential to obtain good fixed-parameter algorithms. Indeed, kernelization has been recognized as one of the key techniques of parameterized algorithmics [1,11,14]. In this context, one of our main contributions is to provide a set of new polynomial-time data reduction rules for 2LP. Due to the lack of space, most proofs are deferred to a long version of this paper.

Preliminaries. Let G be a graph. For every vertex set $V' \subseteq V(G)$, we denote the subgraph of G that is induced by V' by $G[V']$ and we write $G - V'$ for $G[V(G) \setminus V']$. Equivalently, for every edge-set $S \subseteq E(G)$, consider $G - S$ an abbreviation for $(V(G), E(G) \setminus S)$ and $V(S)$ the set of endpoints of edges in S. We denote the neighborhood of a vertex $v \in V(G)$ in G with $N_G(v)$ and the degree of v in G with $\deg_G(v)$. If clear from the context, we omit the index. Furthermore, let $I(G)$ (isolated vertices) and $L(G)$ (leaves) denote the set of vertices in G with degree zero and one, respectively. Following [5], we define the *non-leaf degree* $\widehat{\deg}_G(v) := |N_G(v) \setminus L(G)|$ for every vertex $v \in V(G)$.

A *caterpillar tree* (or caterpillar for short) is a tree where every vertex has non-leaf degree at most two. Equivalently, a caterpillar is a tree that does not contain a 2-claw [7] (see Figure 1*a*)). Thus, caterpillars have a forbidden subgraph characterization. A leaf $v \in L(G)$ is called *critical* if its only neighboring vertex has non-leaf degree two. The definition of critical vertices is motivated by the observation that being a caterpillar is invariant with respect to adding neighbors to non-critical vertices.

Informally speaking, G^* denotes the subgraph of G that contains all edges that are contained in a cycle or that connect vertices that are contained in a cycle. Formally, set $G^0 := G$ and recursively define $G^{i+1} := G^i - (L(G_i) \cup I(G_i))$. Finally, let G^* denote the graph G^i with minimum i such that $G^i = G^{i+1}$. Note that G^* is the empty graph iff G is acyclic (a forest). Moreover, for G being a forest of caterpillar trees, G^1 is a forest of paths. Furthermore, note that $G - V(G^*)$ is acyclic (a forest). For a vertex $v \in V(G^*)$ let T^v denote the tree of $G \setminus E(G^*)$ that is rooted at v. The tree T^v is called the pendant tree of v and v is called its connection point. Furthermore, for a rooted tree T and for a vertex $x \in T$ let T_x denote the subtree of T rooted at x.

The following special pendant trees are of particular interest in this work. For a vertex v, let $L(v) := L(G) \cap N(v)$. A path $p = (\{v, w\}, \{w, x\})$ is called a P_2 with connection point v if $\deg(v) \geq 2$, $\deg(w) = 2$, and $\deg(x) = 1$, see Figure 1*b*) for an example. Vertex w is called the middle point and we refer to it as $m(p)$ and vertex x is called the leaf of p denoted by $l(p)$. For a vertex v let $\mathcal{P}_2(v)$ denote the set of all P_2's that have v as their connection point. A Y-graph is defined as shown in Figure 1*c*). Vertex v is called the connection point and vertex w is called the center point of the Y-graph. We refer to w by $c(Y)$. Let $\mathcal{Y}(v)$ denote the set of all Y-graphs that have v as their connection point.

Our results are in the context of parameterized complexity, which is a two-dimensional framework for studying computational complexity [4,9,14]. One dimension is the input size n, and the other one is the *parameter* (usually a positive integer). A problem is called *fixed-parameter tractable* (fpt) with respect to a

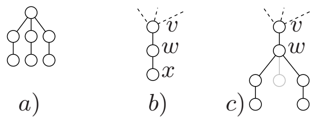

Fig. 1. Terminology. *a*) A 2-claw. The degree-three vertex of a 2-claw is called its center vertex. Caterpillars can be characterized as graphs containing neither cycles nor 2-claws. Figure *b*) shows a P_2 and Figure *c*) a Y-graph. In *c*) the gray leaf may or may not be present in the Y-graph (formally, there are two different Y-graphs, one with and one without the gray leaf).

parameter k if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function only depending on k. A core tool in the development of fixed-parameter algorithms is polynomial-time preprocessing by *data reduction*. Here, the goal is to transform a given problem instance x with parameter k into an equivalent instance x' with parameter $k' \leq k$ such that the size of x' is upper-bounded by some function only depending on k. This is usually achieved by applying data reduction rules. We call a data reduction rule *correct* if the new instance after an application of this rule is a yes-instance iff the original instance is a yes-instance. An instance is called *reduced* with respect to some data reduction rule if this rule can not be applied to the instance. The whole process is called *kernelization*.

2 Kernelizing 2-Layer Planarization

In this section, we present a kernelization for 2LP parameterized by f, denoting the size of a minimum feedback edge set. We present a number of polynomial-time executable data reduction rules and show that a graph that is reduced with respect to these rules cannot contain more than $O(f)$ vertices and $O(f)$ edges. As noted before, this result improves previous work by Dujmović et al. [5]. The kernelization consists of two phases. In the first phase, which we call "tree reduction", roughly speaking, the goal is to reduce the "acyclic part" of the input graph. In the second phase, the goal is to reduce the long non-branching paths in the remaining "cyclic core" G^*, shrinking its size to a function linear in f. We call the second phase "path replacement".

Tree Reduction. Subsequently, we present reduction rules for repeatedly replacing a pendant tree T^u for some $u \in V(G)$ with a smaller tree, until its size is a constant value (see Figure 2 for an illustration). Tree Reduction Rule 1 below is from [5].

Tree Reduction Rule 1 *If there is a vertex v in T^u with $|L(v)| \geq 2$, then delete all but one leaf in $L(v)$.*

Tree Reduction Rule 2 *If there is a vertex v in T^u with $|\mathcal{Y}(v)| \geq 1$ and $|\mathcal{Y}(v)| + |L(v)| + |\mathcal{P}_2(v)| \geq 2$, then, for an arbitrarily chosen Y-graph $Y \in \mathcal{Y}(v)$, delete all vertices of Y except for v and decrease k by one.*

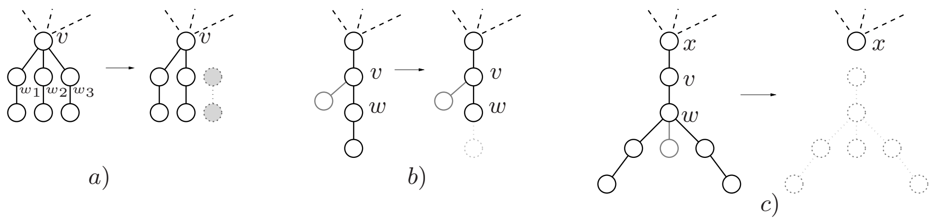

Fig. 2. *a*) Example for the application of Tree Reduction Rule 3. Note that we must delete one of the edges $\{v, w_i\}$, $i = 1, 2, 3$ in order to destroy the 2-claw centered at v. By symmetry, there is an optimal solution that contains the edge $\{v, w_3\}$. *b*) Example for the application of Tree Reduction Rule 4. Note that the edge $\{w, x\}$ is not contained in any 2-claw and hence, x can be deleted. *c*) Example for the application of Tree Reduction Rule 5. Since the deletion of the edge $\{x, v\}$ destroys the same 2-claws as the deletion of any other edge in the tree rooted at x, there is an optimal solution that contains $\{x, v\}$.

Tree Reduction Rule 3 *Consider a vertex v in T^u with $|\mathcal{P}_2(v)| \geq 3$. Let $\mathcal{P}_2(v) = \{p_1, p_2, \ldots, p_q\}$. Delete the vertices $l(p_i)$ and $m(p_i)$ for $3 \leq i \leq q$ and decrease k by $q - 2$.*

Tree Reduction Rule 4 *Consider a vertex v in T^u with $\widehat{\deg}_{T^u}(v) = 2$ and $|\mathcal{P}_2(v)| = 1$. Let $\mathcal{P}_2(v) = \{p\}$. Then delete the vertex $l(p)$.*

Tree Reduction Rule 5 *Consider a vertex v in T^u with $\deg_G(v) = 2$ and $|\mathcal{Y}(v)| = 1$. Let $\mathcal{Y}(v) = \{Y\}$. Then delete all vertices of Y (including v) and decrease k by one.*

Tree Reduction Rule 6 *If C is a connected component of G that is a caterpillar, then delete all vertices of C.*

Lemma 1. *Tree Reduction Rules 1–6 are correct. An instance reduced with respect to Tree Reduction Rules 1–6 can be computed in $O(|V| + |E|)$ time.*

The structure of an instance reduced with respect to these rules is described by the following lemma, which concludes the presentation of the "tree reduction".

Lemma 2. *In a reduced instance, for every vertex $v \in V(G^*)$, its pendant tree T^v is either a singleton or isomorphic to one of the trees shown in Figure 3.*

Path Replacement. The tree reduction rules presented in the previous paragraph are not sufficient to yield a problem kernel for our parameterization. For example, if the input graph G is a simple cycle, then none of the above data reduction rules applies. Recall the notions of G^* (also called the "cyclic core" of G) and pendant trees. The purpose of the subsequently presented data reduction rules is to reduce non-branching paths of G^* (hence, they are called "path reduction

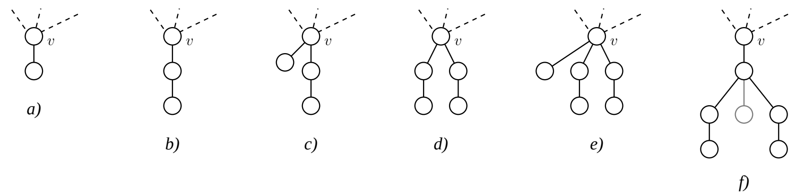

Fig. 3. In a reduced instance, the pendant tree T^v of a vertex $v \in V(G^*)$ is isomorphic to one of the trees shown in a) to f). Note that the tree shown in f) is exactly the Y-graph as defined in Figure 1b).

rules"). The first two reduction rules take care of paths containing Y-graphs as pendant trees. Then, we introduce the notion of tokens which allows us to handle the remaining cases in a unified manner. The reduction rules in this paragraph are more intricate than in the previous paragraph.

In the following, we assume the input to be reduced with respect to all tree reduction rules (see previous paragraph). Consider vertices $u, w \in V(G^*)$ that may be identical. We denote a path $P_{u,w} = (u = v_0, v_1, \ldots, v_\ell, v_{\ell+1} = w)$ between u and w as *degree-2 path* if $\deg_{G^*}(v_i) = 2$ for all $1 \leq i \leq l$. Its *length* is $\ell+2$. We refer to the vertices v_i, $1 \leq i \leq l$, as *inner path vertices*. Furthermore, denote the edges $\{v_{i-1}, v_i\}$ by e_i for all $1 \leq i \leq \ell+1$. Throughout this paragraph, let P be some degree-2 path in the given graph G and let v_i be some inner path vertex of P. In this context, let $T_P := G[\bigcup_{i=1}^{l} V(T^{v_i}) \cup \{u, w\}]$ and for $1 \leq i \leq j \leq l$, let $T_{i,j}$ denote the subtree of T_P containing all vertices reachable from v_i in $T_P - \{e_i, e_{j+1}\}$. Note that $T_{i,i} = T^{v_i}$.

The next two data reduction rules handle all Y-graphs that have a vertex on a degree-2 path as their connection point. First, observe that for any Y-graph Y, deleting the edge that is incident to the connection point is at least as good as deleting any combination of edges of Y.

Observation 1 *Let Y denote some Y-graph in G with connection point v. Then there is an optimal solution S for G with $S \cap E(Y) \subseteq \{\{v, c(Y)\}\}$.*

The first rule identifies Y-graphs Y with connection point v_i for which it is optimal to delete the edge $\{v_i, c(Y)\}$.

Path Reduction Rule 1 *Let P denote a degree-2 path and let both v_i and v_{i+1} be inner path vertices of P such that $Y := T^{v_i}$ is a Y-graph. If*
1. *$T^{v_{i+1}}$ is neither a singleton nor a Y-graph, or*
2. *$\deg_G(v_{i+1}) = 2$, v_{i+2} is an inner path vertex, and $T^{v_{i+2}}$ is either a singleton or a Y-graph,*

then delete $\{v_i, c(y)\}$ and decrease k by one.

The second rule handles almost all remaining cases where a Y-graph occurs as the pendant tree of some inner path vertex v_i of P by bypassing v_i in P.

Path Reduction Rule 2 *Let P denote a degree-2 path and let both v_i and v_{i+1} be inner path vertices of P such that $Y := T^{v_i}$ is a Y-graph. If*

1. $T^{v_{i+1}}$ *is a Y-graph, or*
2. v_{i-1} *and* v_{i+1} *have degree two, or*
3. v_{i+2} *is an inner path vertex,* $\deg(v_{i+1}) = 2$*, and* $T^{v_{i+2}}$ *is neither a singleton nor a Y-graph,*

then remove all vertices of Y *from* G*, insert the edge* $e = \{v_{i-1}, v_{i+1}\}$*, and decrease* k *by one.*

For the correctness of Path Reduction Rule 2, we need the following lemma.

Lemma 3. *Let* v *be a degree-2 vertex in* G^* *such that* T^v *is neither a singleton nor a Y-graph. Then there is an optimal solution* S *for* G *such that* v *is non-critical in* $G - S$*.*

Lemma 4. *Path Reduction Rule 2 is correct.*

Proof. Let G' denote the graph that results from applying Path Reduction Rule 2 to some v_i in G and let $e_Y := \{v_i, c(Y)\}$. For the correctness, we prove that G has a solution S of size k if and only if G' has a solution S' of size $k - 1$.

Let $\hat{G}$ denote the result of contracting e_{i+1} in $G - \{e_Y\}$ and observe that $\hat{G}$ is identical to G' with the exception of one connected component (containing all vertices of Y but v_i) which is a caterpillar. Obviously, $\hat{G}$ and G' are equivalent in the sense that a solution for one is also a solution for the other (considering that e_i in $\hat{G}$ plays the role of e in G').

"$\Rightarrow$:" If $e_Y \in S$, then, since contracting an edge does not create 2-claws or cycles, $S' := S \setminus \{e_Y\}$ is a solution of size $k - 1$ for $\hat{G}$ and, thus, for G'. Otherwise, by Observation 1, no edge of Y is in S and thus, $e_i \in S$ and $e_{i+1} \in S$. Moreover, by construction, $G' - \{e\}$ is a subgraph of $G - \{e_i, e_{i+1}\}$ and thus, $S' := S \setminus \{e_1, e_2\} \cup \{e\}$ is a solution for G' of size $k - 1$.

"$\Leftarrow$:" First, if a solution S' for G' of size $k-1$ contains e, then the equivalent solution $\hat{S}$ for $\hat{G}$ contains e_i, and clearly, $S := \hat{S} \cup \{e_{i+1}\}$ is a size-k solution for G. Thus, in the following, we assume that there is no solution for G' of size $k - 1$ that contains e (in particular, $e \notin S'$ and thus v_{i-1} and v_{i+1} are neighbors in $G' - S'$).

Second, observe that the subdivision of an edge e' of a caterpillar can only create a 2-claw if e' is incident to a critical vertex. Hence, if v_{i-1} and v_{i+1} are both non-critical in $G' - S'$, then we can subdivide e without affecting the solution and thus, $S := S' \cup \{e_Y\}$ is a size-k solution for G. Hence, in the following, we assume that v_{i-1} or v_{i+1} is critical in $G' - S'$. In the following, we consider the three cases of Path Reduction Rule 2 separately.

Case 1: The first condition of Path Reduction Rule 2 applies.
Then, $Y' := T^{v_{i+1}}$ is a Y-graph. Let $e_{Y'} := \{v_{i+1}, c(Y')\}$. By Observation 1, we can assume that S' contains either $e_{Y'}$ or both e and e_{i+2}. However, by the above assumption, $e \notin S'$ and thus, $e_{i+2} \notin S'$ and $e_{Y'} \in S'$, implying $\deg_{G'-S'}(v_{i+1}) = 2$. Thus, neither v_{i+1}, nor v_{i-1} is critical in $G'-S'$, contradicting the assumption above.

Case 2: The second condition of Path Reduction Rule 2 applies.
Then, $\deg_G(v_{i-1}) = \deg_G(v_{i+1}) = 2$. Clearly, if any of v_{i-1}, v_{i+1} is a leaf in $G' -$

S', then the other cannot have non-leaf degree two in $G' - S'$. Thus, neither of them is critical in $G' - S'$, contradicting the assumption above.

Case 3: The third condition of Path Reduction Rule 2 applies.
By Lemma 3 we can assume that v_{i+2} is non-critical in $G' - S'$. Clearly, v_{i-1} is non-critical in $G' - S'$ since $\deg_G(v_{i+1}) = 2$. Hence, v_{i+1} is critical in $G' - S'$ and thus, $S' \cup \{e\}$ isolates v_{i+1}. Since v_{i+2} is non-critical in $G' - S'$, it follows that $(S' \cup \{e\}) \setminus \{e_{i+2}\}$ is a size-k solution for G' containing e, contradicting the assumption above. $\square$

The two path reduction rules presented so far eliminate Y-graphs in all long degree-2 paths. In the following, consider P to be *Y-graph-free*, that is, P does not contain an inner path vertex whose pendant tree is a Y-graph. Consider a graph that is reduced with respect to Path Reduction Rules 1 and 2. All degree-2 paths P' that are not Y-graph-free contain at most two inner path vertices, whose pendant trees are a singleton and a Y-graph, respectively. Thus, it is clear that $|V(T_{P'})| \leq 9$. In the following, we focus on Y-graph-free degree-2 paths. In this case, we can show that we do not need to consider deleting edges in pendant trees, thus allowing us to restrict our attention to deleting edges on the degree-2 path.

Lemma 5. *Let P be a Y-graph-free degree-2 path. There is an optimal solution for G that does not contain any edge of $E(T_P) \setminus E(P)$.*

To handle the remaining paths in a unified manner, we introduce the notion of "tokens" as sets of consecutive edges of P. Consider a vertex v in P that is the center of a 2-claw. Then, Lemma 5 tells us that this 2-claw must be destroyed by deleting an edge of P. We model this fact by letting v generate a token containing all edges that may be deleted in order to destroy this 2-claw. The introduction of this notion is split into three parts. First, we define tokens, second, we point out how they are generated, and third, we specify what it means to destroy a token.

In the following, a vertex v in P is called *crossable* if $\deg_G(v) = 2$. A *token* K of P is a set of at most 4 consecutive edges of P. Let v_i be a vertex in P. If $i \leq \ell - 1$, then the *upper token* $K^{\mathrm{up}}(v_i)$ of v_i is $\{e_{i+1}, e_{i+2}\}$ if v_{i+1} is crossable and it is $\{e_{i+1}\}$, otherwise. Equivalently, if $i \geq 2$, then the *lower token* $K^{\mathrm{low}}(v_i)$ of v_i is $\{e_{i-1}, e_i\}$ if v_{i-1} is crossable and it is $\{e_i\}$ otherwise. In this sense, we say that tokens can only "span" over crossable vertices. For the vertices u, v_1, v_ℓ, and w, we need the following auxiliary tokens: $K^{\mathrm{low}}(u) := K^{\mathrm{up}}(w) := \{\diamond\}$, $K^{\mathrm{low}}(v_1) := \{\diamond, e_1\}$, and $K^{\mathrm{up}}(v_\ell) := \{e_{\ell+1}, \diamond\}$.

Each inner path vertex v_i of P for which $\mathcal{P}_2(v_i) \neq \emptyset$ *generates* tokens in the following way: If $|\mathcal{P}_2(v_i)| = 1$ (in this case T^{v_i} is isomorphic to one of the trees shown in Figure 3 *b*) and *c*)), then v_i generates one token $K^{\mathrm{up}}(v_i) \cup K^{\mathrm{low}}(v_i)$. If $|\mathcal{P}_2(v_i)| = 2$ (in this case T^{v_i} is isomorphic to one of the trees shown in Figure 3 *d*) and *e*)), then v_i generates two tokens, $K^{\mathrm{up}}(v_i)$ and $K^{\mathrm{low}}(v_i)$. We define $\mathcal{K}(v_i)$ as the set of tokens generated by v_i and $\mathcal{K}(P)$ as the set of all tokens generated by inner path vertices of P. See Figure 4 for an illustration.

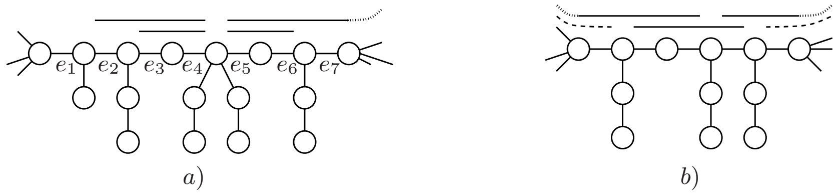

Fig. 4. *a*) Illustration of the tokens generated by a degree-2 path. The tokens are depicted by horizontal bars. That is, the tokens are $\{e_2, e_3, e_4\}$, $\{e_3, e_4\}$, $\{e_5, e_6\}$, and $\{e_5, e_6, e_7, \diamond\}$. *b*) Example of a chained degree-2 path. The tokens are depicted by horizontal bars. Furthermore, dashed lines represent auxiliary tokens.

A non-auxiliary token (token that does not contain $\diamond$) $K \in \mathcal{K}(P)$ is *destroyed* by an edge set E' if $K \cap E' \neq \emptyset$. Observe that if K contains $\diamond$, then it either contains e_1 or e_ℓ. We say that a token containing e_1 and $\diamond$ is destroyed by E' if either $K \cap E' \neq \emptyset$ or E' contains all edges incident to v_0 except for e_1. A token containing $e_{\ell+1}$ is destroyed analogously. Informally speaking, a token represents the need to delete an edge. By Lemma 5 it suffices to consider edges in P. Thus, the task is to destroy all tokens by deleting only few edges.

In the following, we present path reduction rules to shrink degree-2 paths. The first rule reduces degree-2 paths that do not contain any tokens.

Path Reduction Rule 3 *If there is a degree-2 path P with $|V(T_P)| > 7$ and $\mathcal{K}(P) = \emptyset$, then contract $T_{2,\ell-1}$ to a single vertex.*

Next, we concentrate on degree-2 paths generating tokens. Note that for some degree-2 path P, the end vertices u and w could be the center of a 2-claw containing inner path vertices of P. To account for this possibility, we define $\mathcal{K}'(P) := \mathcal{K}(P) \cup \{K^{\text{up}}(u), K^{\text{low}}(w)\}$. Furthermore, since tokens are basically edge sets, they may overlap. This behavior is exploited in the following reduction rules requiring a more formal definition. We call an inner path vertex v_i of P a *token separator* if there is no token in $\mathcal{K}'(P)$ containing both e_i and e_{i+1}. Finally, P is called *chained*, if it does not have a token separator (see Figure 4*b*)).

Path Reduction Rule 4 *Let P be a degree-2 path with $\mathcal{K}(P) \neq \emptyset$ and P is not chained. Let v_i be a token separator of P. Then replace $T_{i,i}$ by two copies of $T_{i,i}$, connect one to v_{i-1} (by inserting an edge between its connection point and v_{i-1}), and connect the other to v_{i+1}.*

See Figure 5 for an illustration of Path Reduction Rule 4.

Path Reduction Rule 5 *Let P be a chained degree-2 path with $\mathcal{K}(P) \neq \emptyset$. Let $M_P := \{i \in \mathbb{N} \mid \mathcal{K}(v_i) \neq \emptyset\}$, let $p := \min M_P$ and $q := \max M_P$, and suppose that $q - p > 1$. If $|\mathcal{K}(P)|$ is even, then delete $T_{p+1,q-1}$, insert the edge $e := \{v_p, v_q\}$, and reduce k by $(|\mathcal{K}(P)| - 2)/2$; otherwise delete $T_{p+1,q}$, insert the edge $e := \{v_p, v_{q+1}\}$, and reduce k by $(|\mathcal{K}(P)| - 1)/2$.*

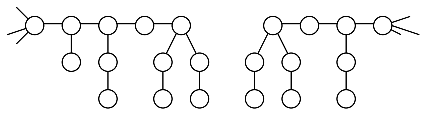

Fig. 5. The graph that results from applying Path Reduction Rule 4 to the graph shown in Figure 4*a*).

Lemma 6. *Let G be reduced with respect to all reduction rules presented in this section and let V_3 denote the set of vertices with degree at least 3 in G^*. If two vertices $u, w \in V_3$ are connected in G by a degree-2 path, then the subgraph of G connecting u and w contains at most 10 vertices and at most 11 edges.*

Theorem 1. 2-Layer Planarization *admits a linear problem kernel containing at most $44(f-1)$ vertices and at most $45(f-1)$ edges, with $f > 0$ being the size of a minimum feedback edge set of G. The problem kernel can be constructed in $O(f \cdot |E|)$ time.*

Proof. Assume that the input G is reduced with respect to all presented data reduction rules. For the analysis of the kernel size, we need the following notation. Let V_3 denote the set of vertices with degree at least 3 in G^* and let $G_3^* := (V_3, E_3)$ denote the mulitgraph on V_3 that contains an edge for every maximal degree-2 path in G. More specifically, E_3 contains an edge $\{u, w\}$ for every edge $\{u, w\} \in E$ with $u, w \in V_3$, and, in addition, E_3 contains an edge $\{u, w\}$ for every maximal degree-2 path of length at least three between two (not necessarily different) vertices $u, w \in V_3$. Thus, G_3^* may contain loops. Furthermore, let F with $|F| = f$ be a minimum feedback edge set of G and let F_3 be a minimum feedback edge set of G_3^* (we require that a feedback edge set of G_3^* contains all loops and all but at most one edge between any two vertices). Clearly, $|F_3| \leq f$ and $G_3^* - F_3$ is a forest and, thus, $|E_3| \leq |V_3| + f - 1$. Since the minimum degree[1] of a vertex in G_3^* is 3, we know that $\sum_{v \in V_3} \deg_{G_3^*}(v) \geq 3|V_3|$, and since the sum on the left hand side equals $2|E_3|$, we know that $2(|V_3| + f - 1) \geq 3|V_3|$, implying $|V_3| \leq 2(f-1)$ and $|E_3| \leq 3(f-1)$.

With G_3^* bounded, we can use Lemma 6 to bound G. Each edge in G_3^* corresponds to a degree-2 path in G^*. Each vertex in V_3 may additionally be incident to a pendant tree (see Figure 3). Thus, we can bound the number of vertices in G by $|V(G)| \leq |E_3| \cdot 10 + |V_3| \cdot 6 + |V_3| \leq 44(f-1)$ and the number of edges in G by $|E(G)| \leq 45(f-1)$. $\square$

3 Search Tree, Further Results, and Open Questions

In this section, we provide an algorithm that solves the 2-Layer Planarization problem in $O(6^f \cdot f^2 + f \cdot |E|)$ time. It employs a search tree based strategy

[1] A loop at vertex v contributes 2 to the degree of v.

that makes use of the kernelization presented in the previous section. The algorithm runs in three phases. First, we apply a search tree enumerating partial solutions by branching on a certain type of 2-claw. Second, we branch on small cycles in the remaining graph. In the third phase, we branch on the tokens (see Section 2, page 8) that remain in the graph. The input graph G considered in each phase is assumed not to be subject to the previous phase, that is, G does not contain a structure that is branched on in the previous phase. Furthermore, the input graph of each phase is assumed to be reduced with respect to the data reduction rules presented in the previous section. We split the number of edge deletions done by branching in the three phases into f_1, f_2, and f_3 with $f_1 + f_2 + f_3 = f$.

Theorem 2. 2-LAYER PLANARIZATION *can be solved in* $O(6^f \cdot f \cdot |E|)$ *time.*

By initially kernelizing the input instance, we can assume $|E|$ to be linear in f for the branching algorithm.

Corollary 1. 2-LAYER PLANARIZATION *can be solved in* $O(6^f \cdot f^2 + f \cdot |E|)$ *time. Moreover, if* $|E| \leq |V| + O(\log |V|)$*, then* 2-LAYER PLANARIZATION *can be solved in polynomial time.*

We can show that our kernelization results for 2LP can be extended to the closely related NP-hard NODE DUPLICATION BASED CROSSING ELIMINATION(NDCE) problem [2]. In NDCE the task is to transform an input graph into a forest of caterpillar trees by a minimum number of node duplications. Herein, duplicating a vertex v means to delete v, add two new vertices v_1, v_2, and make every former neighbor of v adjacent to exactly one of v_1 or v_2. NDCE arises in the design of molecular quantum-dot cellular automata [2] and visualization of gene ontologies in bioinformatics [18]. Chaudhary et al. [2] showed the NP-hardness of NDCE and presented an ILP formulation for it. The parameterized complexity of NDCE seems unexplored. We can show that NDCE has a linear problem kernel with respect to the feedback edge set number of the graph. Note that the feedback edge set number is also a lower bound for the number of node duplications required to transform a graph into a forest of caterpillars. Since the techniques employed by this kernelization are very similar to the ones presented for 2LP, we defer these elaborations to a long version of this paper.

We conclude with some open questions for future work. It is interesting to investigate whether our kernelization approach also holds for the edge-weighted case. Moreover, it is interesting to investigate whether the branching analysis suggested by Suderman [16] can be used to obtain a better search tree algorithm for the parameter feedback edge set number. Providing efficient fixed-parameter algorithms for parameters upper-bounded by the feedback edge set is a natural next step to extend the range of solvable instances. The feedback vertex set number would be a canonical candidate. Finally, it would be interesting to extend our results to the multilayered problem versions [6].

References

1. H. L. Bodlaender. Kernelization: New upper and lower bound techniques. In *Proc. 4th IWPEC*, volume 5917 of *LNCS*, pages 17–37. Springer, 2009.
2. A. Chaudhary, D. Z. Chen, X. S. Hu, M. T. Niemier, R. Ravichandran, and K. Whitton. Fabricatable interconnect and molecular QCA circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 26(11):1978–1991, 2007.
3. M. Dom, J. Guo, and R. Niedermeier. Approximation and fixed-parameter algorithms for consecutive ones submatrix problems. *Journal of Computer and System Sciences*, 2010.
4. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
5. V. Dujmović, M. Fellows, M. Hallett, M. Kitching, G. L. C. McCartin, N. Nishimura, P. Ragde, F. Rosamond, M. Suderman, S. Whitesides, and D. R. Wood. A fixed-parameter approach to 2-layer planarization. *Algorithmica*, 45(2):159–182, 2006.
6. V. Dujmović, M. Fellows, M. Hallett, M. Kitching, G. L. C. McCartin, N. Nishimura, P. Ragde, F. Rosamond, M. Suderman, S. Whitesides, and D. R. Wood. On the parameterized complexity of layered graph drawing. *Algorithmica*, 52(2):267–292, 2008.
7. P. Eades and S. Whitesides. Drawing graphs in two layers. *Theoretical Computer Science*, 131(2):361–374, 1994.
8. H. Fernau. Two-layer planarization: Improving on parameterized algorithmics. *Journal of Graph Algorithms and Applications*, 9(2):205–238, 2005.
9. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
10. J. Guo, F. Hüffner, and R. Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proc. 1st IWPEC*, volume 3162 of *LNCS*, pages 162–173. Springer, 2004.
11. J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007.
12. T. Lengauer. *Combinatorial algorithms for integrated circuit layout*. John Wiley & Sons, Inc., New York, NY, USA, 1990.
13. P. Mutzel. An alternative method to crossing minimization on hierarchical graphs. *SIAM Journal on Optimization*, 11(4):1065–1080, 2001.
14. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Number 31 in Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.
15. F. Shahrokhi, O. Sýkora, L. A. Székely, and I. Vrťo. On bipartite drawings and the linear arrangement problem. *SIAM Journal on Computing*, 30(6):1773–1789, 2001.
16. M. Suderman. *Layered Graph Drawing*. PhD thesis, School of Computer Science, McGill University Montréal, 2005.
17. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, 1981.
18. V. Tsiaras, S. Triantafilou, and I. G. Tollis. DAGmaps: Space filling visualization of directed acyclic graphs. *Journal of Graph Algorithms and Applications*, 13(3):319–347, 2009.
19. M. S. Waterman and J. R. Griggs. Interval graphs and maps of DNA. *Bulletin of Mathematical Biology*, 48(2):189–195, 1986.