Dynamic Algorithms for Graphs with Treewidth 2*

Hans L. Bodlaender

Department of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands

Abstract. In this paper, we consider algorithms for maintaining tree-decompositions with constant bounded treewidth under edge and vertex insertions and deletions for graphs with treewidth at most 2 (also called: partial 2-trees, or series-parallel graphs), and for almost trees with parameter k. Each operation can be performed in $O(\log n)$ time. For a large number of graph decision, optimization and counting problems, information can be maintained using $O(\log n)$ time per update, such that queries can be resolved in $O(\log n)$ or $O(1)$ time. Similar results hold for the classes of almost trees with parameter k, for fixed k.

1 Introduction

Two recently popular areas of investigations in graph algorithms are dynamic graph algorithms, and algorithms for graphs with small treewidth. In this paper, we consider dynamic algorithms for graphs with treewidth at most 2, (also known as *partial 2-trees* or *series-parallel graphs.*) These contain all outerplanar graphs.

Many problems become linear time solvable for graphs, given together with a tree-decomposition with treewidth bounded by some constant k. (Such a tree-decomposition can be found in $O(n)$ time [4] (see also [13, 15]).) These problems include many well known NP-complete problems, like Hamiltonian Circuit, Independent Set, Graph Coloring, etc., counting problems like *How many Hamiltonian circuits does G have?* and some PSPACE-complete problems. Also, these problems when restricted to graphs with bounded treewidth belong to NC [3, 7, 12].

We consider the problem of solving these problems on graphs with treewidth at most 2 that change dynamically. We allow the following operations: insertion and deletion of isolated vertices, insertion of edges that do not result in a graph with treewidth larger than 2, and deletion of edges. One can check (using the results of this paper) in $O(\log n)$ time whether a desired edge insertion would yield a graph with treewidth at least 3. Also, when considering problems on weighted or labeled graphs, we allow operations that change the label or weight of a vertex or edge.

For a large class of graph decision, optimization, and counting problems, we show that data structures can be maintained, such that each such operation and queries to the problem can be performed in $O(\log n)$ time. These problems include almost all problems known to be linear time solvable on graphs with bounded treewidth.

Besides graphs with treewidth at most two, similar results also hold for the classes of graphs of *almost trees with parameter k*, for some constant k.

* This work was partially supported by the ESPRIT Basic Research Actions of the EC under contract 7141 (project ALCOM II).

Related results. In a recent paper, Cohen et al [8] designed algorithms for the maintenance of graphs with treewidth at most 2 or 3. For graphs with treewidth at most 2, insertions and deletions can be done in $O(\log^2 n)$ time, while queries cost $O(\log n)$ time. For graphs with treewidth at most 3, their data structure allows insertions (no deletions) to be performed in $O(\log n)$ amortized time. The class of graph problems that can be queried with their approach is much smaller than the class, dealt with in the present paper. The techniques used in [8] and this paper are quite different.

Frederickson [10, 11] found independently similar results for trees, forests, and k-terminal trees under several operations, including label changes. The technique in this paper for maintaining trees and forests is very similar to the technique used in [10, 11].

Fernandez-Baca and Slutzki [9] considered parametrized algorithms on graphs with bounded treewidth.

Overview of this paper. In section 2, definitions and some preliminary results are reviewed. In section 3, we discuss the class of query problems we can deal with. In section 4, we use 'parallel tree-contraction' to come to the data structure and algorithms that maintain suitable tree-decompositions of binary forests. In section 5, we discuss how the result of section 4 can be used to deal with larger classes of graphs, including the graphs with treewidth at most 2, and the almost trees with parameter k. Consequences of these results, open problems, and some other final remarks can be found in section 6.

2 Definitions and preliminary results

The notion of tree-width and tree-decomposition were introduced by Robertson and Seymour in their series of fundamental papers on graph minors[16].

Definition 1. A *tree-decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with $\{X_i \mid i \in I\}$ a collection of subsets of vertices, and T a tree, such that

- $\bigcup_{i \in I} X_i = V$.
- $\forall (v, w) \in E : \exists i \in I : v, w \in X_i$.
- $\forall v \in V : \{i \in I \mid v \in X_i\}$ induced a connected subtree of T.

The treewidth of tree-decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The treewidth of a graph G is the minimum treewidth over all possible tree-decompositions of G.

The third condition can be equivalently be replaced by: for all $i, j, k \in I$, if j lies on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$. There are several notions, equivalent to treewidth, e.g. a graph is a 'partial k-tree', iff its treewidth is at most k (see e.g. [17]). We say a tree-decomposition is *nice*, if T is a rooted binary tree, with for root node r: $X_r = \emptyset$. It is easy to transform each tree-decomposition into a nice one with the same treewidth.

A $\leq k$-boundary graph $G = (V, E, B)$ is a 3-tuple, with (V, E) an undirected graph, and B a set of at most k vertices in V. Consider a nice tree-decomposition

$(\{X_i \mid i \in I\}, T = (I,F))$ of treewidth $k-1$. For $i \in I$, let $Y_i = \{v \in X_j \mid j = i$ or j is a descendant of $i\}$. Write $G[Y_i] = (Y_i, E_i)$. For all $i \in I$, (Y_i, E_i, X_i) is a $\leq k$-terminal graph.

Many linear time algorithms that solve problems on graphs with a given (nice) tree-decomposition can be expressed in the following way (after 'cosmetical changes'):

For each node $i \in I$, some information about (Y_i, E_i, X_i) is computed. Denote this information by $\alpha(i)$. Each value $\alpha(i)$ can be computed in $O(1)$ time when given all values $\alpha(j)$ for all children j of i. Thus, the values can be computed in $O(n)$ time in total, working bottom-up in the decomposition tree. From $\alpha(r)$, the answer to the problem that is to be solved can be determined in $O(1)$ time. In case each $\alpha(i)$ contains only a constant bounded number of bits, we call the problem *finite state* (after Abrahamson and Fellows [1].)

Next, we review some useful results on the structure of graphs with treewidth at most 2. A graph has treewidth at most 1, if and only if it is a forest.

Consider a graph $G = (V,E)$ with treewidth at most 2. Let $H = (V, E')$ be the graph, obtained by adding to G all edges $(v,w) \notin E$ for all pairs $v, w \in V$, $v \neq w$, for which there are at least three vertex disjoint paths between v and w in G. H has also treewidth ≤ 2, and is called the *cell completion* of G. H is the cell completion of itself. We say a graph G is completed, if it is the cell completion of a graph with treewidth at most 2.

A completed graph H has a nice, and relatively easy structure: each biconnected component either consists of a single edge, or can be made in the following way: start with a simple cycle, and then iteratively add zero or more cycles, each new cycle sharing exactly one edge with the part of the component made so far. (This characterization is due to Ton Kloks. See e.g. [5].) We say an edge (v,w) in a completed graph H is a *base edge*, if, besides the edge (v,w), there are two other vertex disjoint paths from v to w in H. (It is an edge where cycles were 'glue-ed together'.)

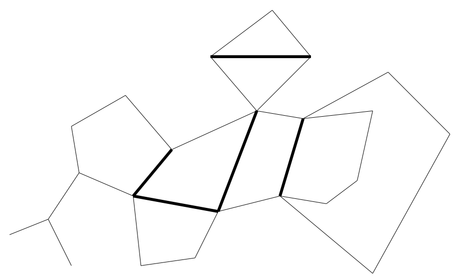

Fig. 1. A completed graph with base edges

A graph $G = (V,E)$ is an *almost tree with parameter k*, if G has a spanning forest $T = (V,F)$, such that each biconnected component of G contains at most k edges not in T. A graph is a *cactus*, if it is an almost tree with parameter 1. Equivalently, if no edge belongs to more than one simple cycle. The treewidth of an almost tree with parameter k is at most $k+1$.

3 Td-open problems

Suppose we have a dynamically changing graph $G=(V,E)$ with a tree-decomposition that may change dynamically too. When solving problems (like Hamiltonian Circuit, Independent Set) on G, it is desirable that only few values $\alpha(i)$ (see section 2) need to be recomputed during a change of G and its tree-decomposition. We will see that for many problems, the update time can made linear in the number of *affected* nodes, where *affected* is defined as follows:

- In case of a change of a weight or a label of a vertex $v \in V$, or an edge $(w,x) \in E$, let $i \in I$ be the highest node in the tree containing v, or containing both w and x, respectively. All predecessors of i (not i itself) are *affected*.
- In case of a change of $G=(V,E)$ to $G'=(V',E')$ and of its tree-decomposition $(\{X_i \mid i \in I\}, T=(I,F))$ to tree-decomposition $(\{X'_i \mid i \in I'\}, T'=(I',F'))$ of G', the following nodes in I' are *affected*:
  - All nodes in I', not in I.
  - All nodes $i \in I \cap I'$ whose subtree of T, rooted at i is not identical to the subtree of T', rooted at i. This includes the case that for some descendant $j \in I \cap I'$ of i, $X_j \neq X'_j$.
  - All nodes $i \in I'$, for which there are vertices $v, w \in V \cap V'$ with $(v,w) \in E - E' \cup E' - E$ and i is a predecessor of the highest node j in T with $v, w \in X_j$.

Note that any predecessor of an affected node is also affected. Graph problems are described as functions, that map graphs $G=(V,E)$ with vertex and edge labelings $lab_V : V \to L_V$, $lab_E : E \to L_E$ to some set of answers.

Definition 2. A graph problem P is *td-open*, if there exists a function α, that maps nodes in nice tree-decompositions of the input graph to a value in some set S, such that

- for each node i, $\alpha(i)$ can be computed in $O(1)$ time, given all values $\alpha(j)$ for all children j of i.
- the desired answer $P(G, lab_V, lab_E)$ can be determined in $O(1)$ time from $\alpha(r)$ (where r is the root node of the decomposition tree).
- when G, lab_V, lab_E and/or the tree-decomposition is changed, then $\alpha(i)$ is not changed for nodes that are not affected by the change.

If P is td-open, then maintaining the necessary information to solve P can be done in time, proportional to the number of affected nodes.

A powerful language to express many linear time solvable problems on graphs with given tree-decomposition of bounded treewidth is the Monadic Second Order Logic (MSOL) and its extensions. See e.g. [2, 6].

Theorem 3. *(i) If a problem P is finite state, then P is td-open.*
(ii) If a problem P is expressible in extended monadic second order logic, then P is td-open.

Proof. (i) The basic idea is: $\alpha(i)$ contains all states (as in the original algorithm) for all possible labelings of X_i, $E[X_i]$, and all possible sets of edges in $X_i \times X_i$. This information is still an element from a finite set, can be computed in $O(1)$ time from the information from the children of i, and does not change under updates that do not affect i.
(ii) By simple modification of the algorithm in [6]. □

Also, counting versions of finite state or MSOL problems (like counting the number of Hamiltonian Circuits) are td-open. Most other problems, solvable in linear time on graphs with given tree-decomposition of bounded treewidth can also shown to be td-open. Many td-open problems are listed in section 6. Clearly, problems that can have output of non-constant size (like: output a set of vertices that is a maximum independent set) cannot be td-open.

4 Maintenance of tree-decompositions of binary trees with logarithmic depth

Definition 4. Let $\mathcal{G}$ be a class of graphs, and let *op* be a set of operations $\mathcal{G} \to \mathcal{G}$. We say that $(\mathcal{G}, op)$ can be *strongly td-maintained* with cost $O(f(n))$, if data-structures and algorithms exist, which maintain a nice tree-decomposition of a dynamically changing (by operations in *op*) graph $G \in \mathcal{G}$, such that

- the depth of the decomposition tree is $O(\log n)$.
- there is a uniform upper bound on the treewidth of the tree-decompositions.
- each operation in *op* can be executed in $O(f(n))$ time, such that there are $O(f(n))$ affected nodes.
- for each vertex $v \in V$, there are at most $O(f(n))$ nodes i with $v \in X_i$ or i a predecessor of a node j with $v \in X_j$.

If the last condition does not hold, we say that $(\mathcal{G}, op)$ is *td-maintained* with cost $O(f(n))$.

Theorem 5. *Suppose $(\mathcal{G}, op)$ can be td-maintained with cost $O(f(n))$. For each finite set of td-open problems on (labeled) graphs in $\mathcal{G}$, a data-structure exists, that allows operations in op, label changes of a vertex and/or edge, and queries to the problems to be executed in $O(f(n))$ time.*

Usually, queries cost even only $O(1)$ time. In this section we show that binary forests with deletions, and insertions of and isolated vertices (and several other 'local' operations) can be td-maintained with cost $O(\log n)$.

We use the notations $dele_{\mathcal{G}}$, $inse_{\mathcal{G}}$, $delv_{\mathcal{G}}$, $insv_{\mathcal{G}}$, $compr_{\mathcal{G}}$, $contr_{\mathcal{G}}$, $subdiv_{\mathcal{G}}$ for the operations: delete an edge, insert an edge, delete an isolated vertex, insert an isolated vertex, compress over a vertex of degree 2 (remove the vertex and connect its two old neighbors), contract over an edge, subdivide an edge, in each case provided that the resulting graph belongs to $\mathcal{G}$. For a class of graphs $\mathcal{G}$, denote $OP^-_{\mathcal{G}} = \{dele_{\mathcal{G}}$, $inse_{\mathcal{G}}$, $delv_{\mathcal{G}}$, $insv_{\mathcal{G}}$, $compr_{\mathcal{G}}$, $subdiv_{\mathcal{G}}\}$, and denote $OP_{\mathcal{G}} = OP^-_{\mathcal{G}} \cup \{contr_{\mathcal{G}}\}$.

We base our approach on applying parallel tree-contraction, as introduced by Miller and Reif [14]. In [3] tree-contraction was used in the first proof of membership in NC of many problems on graphs with constant bounded treewidth. (Later results, especially those of Lagergren [12] gave very large savings in the number of used processors.) We use a version of tree-contraction that is most suitable for our purposes.

Define the following operations on forest T with maximum vertex degree 3.

- Rake. Remove a leaf node v, and its adjacent edge (v, w). We say that v and (v, w) are *involved* in this operation and that w represents v and (v, w) after the rake.
- Compress. Take a node v with degree 2. Let w and x be the neighbors of v. Replace v, (v, w), and (v, x) by an edge (w, x). We say that v, (v, w), and (v, x) are *involved* in the operation and that (w, x) *represents* v, (v, w), (v, x).
- 0-remove. Remove an isolated vertex v from T. v is *involved* in this operation.

A set of rake, compress and 0-remove operations is said to be a *good RC-set*, if no vertex or edge is involved in more than one operation, and it is maximal, i.e., every vertex of degree 0 is 0-removed, and every vertex of degree 1 or 2 is adjacent to an edge that is involved.

A contraction series of a forest $T = (V, E)$ is a sequence $(T_0,\ S_0, T_1,\ S_1, \ldots, T_{r-1}$, $S_{r-1},\ T_r)$ with $T_0 = T$, T_r the empty graph, and each T_i $(i \geq 1)$ is the forest, obtained by applying the good RC-set S_{i-1} to forest T_{i-1}. Write $T_i = (V_i, E_i)$.

Lemma 6. *The length r of a contraction series $(T_0,\ S_0,\ T_1, \ldots, T_r)$ of a forest $T = (V, F)$ is $O(\log |V|)$.*

Proof. Similar as in [14]. □

Given a contraction series $(T_0,\ S_0,\ T_1,\ \ldots,\ T_r)$ of a binary tree $T = T_0$, we can build a tree-decomposition $(\{X_i \mid i \in I\}, \mathcal{T} = (I, F))$ of T with treewidth 2 with $\mathcal{T}$ a tree of depth r with degree at most 6, in the following way. Let I be the disjoint union of $V_0,\ V_1, \ldots, V_r,\ E_0, E_1, \ldots, E_r$. (We use superscripts to denote to what V_j, E_j a vertex or edge belongs.) If a vertex $v^j \in V_j$ (edge $(v, w)^j \in E_j$) is not involved in an operation in S_j, then v^{j+1} $((v, w)^{j+1})$ has one child, namely v^j $((v, w)^{j+1})$, and $X_{v^{j+1}} = \{v\}$ $(X_{(v,w)^{j+1}} = \{v, w\})$. If a rake on v^j, $(v, w)^j$ is done in S_j, then the children of w^{j+1} are those vertices and edges in $V_j \cup E_j$ that are represented by w (at most two vertices and two edges, if both children of w are raked). $X_{w^{j+1}}$ contains w and v, and possibly the other child of w if that is also raked in S_j. If a compress on v, (v, w), (v, x) is done, then (w, x) has three children: v^j, $(v, w)^j$, and $(v, x)^{j+1}$.

Using the modification described in section 2, $\mathcal{T}$ can be made binary, while its depth increases with only a constant factor. Extending this technique to binary forests can be done without much problems by building for each tree a tree-decomposition as above, and then using extra nodes i with $X_i = \emptyset$, which are at the top of the resulting tree and together have logarithmic depth.

Let $\mathcal{F}_3$ denote the set of all forests with maximum vertex degree 3. The reason why the construction described above is useful for dynamic algorithms is the following theorem, which can be shown by an extensive case analysis.

Theorem 7. *There exists a finite set X of operations $\mathcal{O}$, each involving a constant bounded number of vertices and edges, which contains all operations in $OP_{\mathcal{F}_3}$, such that for every forest $T_0 = (V_0, E_0)$, for every forest $T_0' = (V_0', E_0')$ obtained by applying an operation in $\mathcal{O}$ to T_0, and for every forest $T_1 = (V_1, F_1)$ that is obtained by applying a good RC-set S to T_0, there exists a good RC-set S' on T_0', such that the forest T_1' obtained by applying S' to T_0' can also be obtained by applying one operation from $\mathcal{O}$ to T_1.*

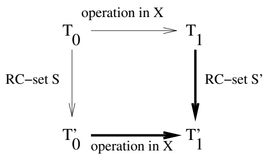

Fig. 2. Graphical representation of theorem 7

Theorem 8. *$(\mathcal{F}_3, OP_{\mathcal{F}_3})$ can be strongly td-maintained with cost $O(\log n)$.*

Proof. Maintain a contraction series of the forest, making repetive use of theorem 7. Changes in the contraction series are directly reflected in the corresponding tree-decompositions. A 'small change' in the forest will affect only a constant bounded vertices and edges per forest in the contraction series, hence $O(\log n)$ nodes will be affection in total with an operation. Strongness follows as each vertex belongs to $O(1)$ nodes per level of the decomposition tree. □

5 Larger classes of graphs

In this section, we show for larger classes of graphs that they can be td-maintained (with the usual operations) with $O(\log n)$ cost.

First, we consider the class $\mathcal{CAC}_3$ of the cactus graphs with maximum vertex degree 3. We maintain a maximal spanning forest **T**I= (V, E') of cactus $G = (V, E)$. If we have a nice tree-decomposition $(\{X_i \mid i \in I\}, \mathcal{T} = (I, F))$ of **T** with treewidth 2, we can make a nice tree-decomposition of G in the following way: for each edge not in the spanning forest $(v, w) \in E - E'$, add either v or w to all nodes in I on the path in $\mathcal{T}$ between the highest node that contains v and the highest node that contains w (inclusive). One can show that this gives a tree-decomposition of G with treewidth at most 5. Simple analysis of the different cases show that each insertion or deletion can be done in $O(\log n)$ time, such that also $O(\log n)$ nodes are affected. As each other operation in $OP_{\mathcal{CAC}_3}$ can be expressed in $O(1)$ insertions and/or deletions, we have:

Lemma 9. *$(\mathcal{CAC}_3, OP_{\mathcal{CAC}_3})$ can be td-maintained with cost $O(\log n)$.*

The main technique to come to larger classes of graphs is 'interpreting' a graph into another graph from a 'simpler' class of graphs.

Definition 10. An *interpretation* of a graph $G = (V_G, E_G)$ into a graph $H = (V_H, E_H)$ is a function $f : V_G \to \mathcal{P}(V_H)$, mapping each vertex $v \in V_G$ to a set of vertices $\subseteq V_H$, such that for all $v \in V$, $f(v)$ induces a connected subgraph in H, and for all $(v, w) \in E_G$: there exist $x \in f(v)$, $y \in f(w)$ with $x = y$ or $(x, y) \in E_H$.

The *inverse* of interpretation f is a pair $(f^{-1,V}, f^{-1,E})$ with $f^{-1,V} : V_H \to \mathcal{P}(V_G)$ defined by $f^{-1,V}(v) = \{w \in V_G \mid v \in f(w)\}$, and $f^{-1,E} : E_H \to \mathcal{P}(E_G)$ defined by $f^{-1,E}((x, y)) = \{(v, w) \in E_G \mid x \in f(v) \text{ and } y \in f(w)\}$.

The *width* of interpretation f is $w(f) = \max_{x \in V_H} |f^{-1,V}(x)|$. The *breadth* of interpretation f is $b(f) = \max_{v \in V_G} |f(v)|$.

Lemma 11. *Let $(\{X_i \mid i \in I\}, T = (I, F))$ be a tree-decomposition of $H = (V_H, E_H)$ with treewidth k, and let $f : V_G \to V_H$ be an interpretation of $G = (V_G, E_G)$ into H. Then $(\{Y_i \mid i \in I\}, T = (I, F))$ with $Y_i = \bigcup_{x \in X_i} f^{-1,V}(x)$ is a tree-decomposition of G with treewidth at most $w(f) \cdot (k + 1) - 1$.*

We say that a class of graphs $\mathcal{G}$ with set of operations $\mathcal{G} \to \mathcal{G}$, $op_{\mathcal{G}}$ can be *interpreted* with width c (and breadth b) into a class of graphs $\mathcal{H}$ with set of operations $\mathcal{H} \to \mathcal{H}$, $op_{\mathcal{H}}$, if for each $G \in \mathcal{G}$, we have a non-empty collection C_G of pairs (H, f), with $H \in \mathcal{H}$, and f an interpretation of G into H with width at most c (and breadth at most b), such that for all $G \in \mathcal{G}$, $(H, f) \in C_G$, and $G' \in \mathcal{G}$ obtained by applying one operation from op_{calG} to G, there exists a sequence *opseq* of operations in $op_{\mathcal{H}}$ with its length bounded by some constant, such that when the operations in *opseq* are sequentially applied to H, a graph H' results, and there exists an interpretation f' of G' into H with $(H', f') \in C_{G'}$. Moreover $f^{-1,V}$ and $(f')^{-1,V}$ must be identical for all vertices in $H \cap H'$ that are not involved in one or more operations in *opseq*, and the time, needed to find the sequence *opseq* must be at most $O(\log n)$, plus the time needed for solving $O(1)$ td-open problems on G and/or on H. (As tree-decompositions of G and H are maintained, the time for solving these td-open problems will be also $O(\log n)$.)

Lemma 12. *Let $\mathcal{G}$, $\mathcal{H}$ be classes of graphs, and $op_{\mathcal{G}}$, $op_{\mathcal{H}}$ be finite sets of operations on graphs in $\mathcal{G}$, $\mathcal{H}$, respectively, where each operation in these classes never involves more than a constant bounded number of vertices and edges.*
(i) If for constants $c, b \in N$, $(\mathcal{G}, op_{\mathcal{G}})$ can be interpreted into $(\mathcal{H}, op_{\mathcal{H}})$ with width c and breadth b, and $(\mathcal{H}, op_{\mathcal{H}})$ can be strongly td-maintained with cost $O(\log n)$, then $(\mathcal{G}, op_{\mathcal{G}})$ can be strongly td-maintained with cost $O(\log n)$.
(ii) If for constant $c \in N$, $(\mathcal{G}, op_{\mathcal{G}})$ can be interpreted into $(\mathcal{H}, op_{\mathcal{H}})$ with width c, and $(\mathcal{H}, op_{\mathcal{H}})$ can be strongly td-maintained with cost $O(\log n)$, then $(\mathcal{G}, op_{\mathcal{G}})$ can be td-maintained with cost $O(\log n)$.

Proof. C_G denotes the possible (graph - interpretation) pairs for G. We maintain one such pair. Then the tree-decomposition of G is made from the tree-decomposition of H as in lemma 5.2. One operation in G translates to $O(1)$ operations to its 'interpreted graph' and interpretation, resulting in $O(\log n)$ changes to the tree-decompositions of G and H. We maintain also the 'α-information' of the td-open problems on G and H, necessary to find the operation sequences *opseq*. $\square$

Let $\mathcal{F}$ ($\mathcal{F}_c$) denote the class of forests (with maximum vertex degree c); let $\mathcal{CAC}$ ($\mathcal{CAC}_c$) denote the class of cactus graphs (with maximum vertex degree c). The following lemma illustrates our technique.

Lemma 13. *$(\mathcal{F}_4, OP_{\mathcal{F}_4})$ can be interpreted into $(\mathcal{F}_3, OP_{\mathcal{F}_3})$ with width 1 and breadth 2.*

Proof. Each forest $T = (V, F) \in \mathcal{F}_4$ is interpreted into forests T', where every vertex of degree 4 in T is replaced by two adjacent vertices of degree 3. See for an example figure 3. Adding an edge (v, w) to T can be done as follows: if the degree of v is

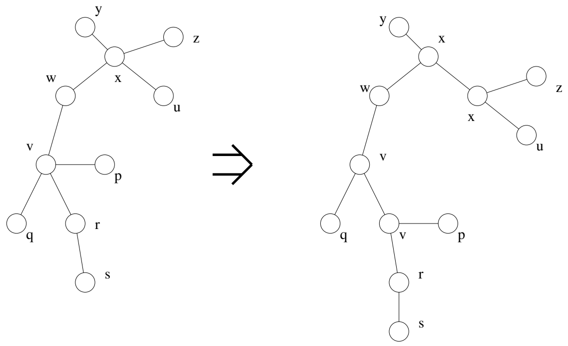

Fig. 3. Interpreting a tree with degree 4 into a tree with degree 3

3, then subdivide one edge adjacent to $f(v)$, put the new vertex also in $f(v)$, and attach the edge to be added to the new vertex. Similar for $f(w)$. Thus, the operation sequence consists of at most two subdivisions and one edge addition.

When deleting an edge (v, w) from T, first delete its counterpart, say (x, y) from T'. Check whether x now has degree 2 and is adjacent to a vertex z with $f^{-1,V}(x) = f^{-1,V}(z)$. If so, compress over x. Do the same for y.

The other operations are straight-forward. $\square$

It follows that $(\mathcal{F}_4, OP_{\mathcal{F}_4})$ can be strongly td-maintained. We give many similar results without proof, or with only an indication of the used interpretations.

Lemma 14. *(i) For all $c \geq 4$, $(\mathcal{F}_c, OP_{\mathcal{F}_c})$ can be strongly interpreted into $(\mathcal{F}_3, OP_{\mathcal{F}_3})$ with width 1 and breadth $c - 2$.*
(ii) $(\mathcal{F}, OP_{\mathcal{F}} - \{contr_{\mathcal{F}}\})$ can be interpreted into $(\mathcal{F}_3, OP_{\mathcal{F}_c})$ with width 1.
(iii) For all $c \geq 4$, $(\mathcal{CAC}_c, OP_{\mathcal{CAC}_c})$ can be strongly interpreted into $(\mathcal{CAC}_3, OP_{\mathcal{CAC}_3})$ with width 1 and breadth $c - 2$.
(iv) $(\mathcal{CAC}, OP^{-}_{\mathcal{CAC}})$ can be interpreted into $(\mathcal{CAC}_3, OP_{\mathcal{CAC}_3})$ with width 1.

Let $\mathcal{COM}$ ($\mathcal{COM}_c$) be the class of the completed graphs (with maximum vertex degree c). We say a completed graph is *cactaceous*, if no two base edges share a vertex. $\mathcal{COMC}_c$ denotes the cactaceous completed graphs with maximum vertex degree c. $\mathcal{TW}_2$ denotes the class of graphs with treewidth at most 2.

Lemma 15. *$(\mathcal{COMC}_4, OP_{\mathcal{COMC}_4})$ can be interpreted into $(\mathcal{CAC}_6, OP_{\mathcal{CAC}_6})$ with width 2 and breadth 1.*

Proof. If we contract each edge in a cactaceous completed graph, then we obtain a cactus. □

Lemma 16. *(i) $(\mathcal{COM}_4, OP_{\mathcal{COM}_4})$ can be interpreted into $(\mathcal{COMC}_4, OP_{\mathcal{COMC}_4})$ with width 1 and breadth 2.*
(ii) $(\mathcal{COM}, OP^{-}_{\mathcal{COM}})$ can be interpreted into $(\mathcal{COM}_4, OP_{\mathcal{COM}_4})$ with width 1.

Proof. (i) Split a vertex which is adjacent to two base edges. No vertex can be adjacent to more than two base edges in a graph in $\mathcal{COM}_4$. See figure 5.

(ii) Each base edge is replaced by a ladder, as shown in figure 5. Vertices that are adjacent to more than one biconnected component may be split, similar as in lemma 13. □

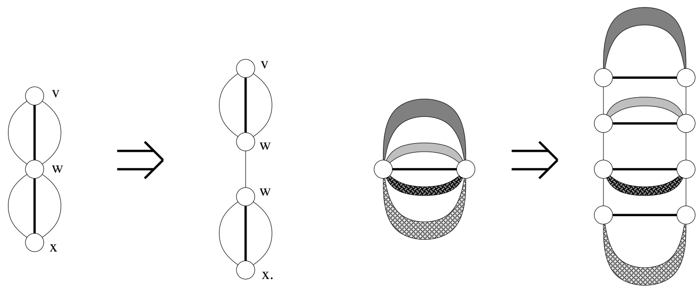

Fig. 4. Interpretations of lemma 16

Lemma 17. *(i) $(\mathcal{TW}_2, OP^{-}_{\mathcal{TW}_2})$ can be interpreted into $(\mathcal{COM}, OP^{-}_{\mathcal{COM}})$ with width 1 and breadth 1.*
(ii) $(\mathcal{TW}_2, OP^{-}_{\mathcal{TW}_2})$ can be interpreted into $(\mathcal{COM}_4, OP_{\mathcal{COM}_4})$ with width 1.

Proof. (i) We interprete a graph G with treewidth ≤ 2 into its completed graph H. E.g., when adding an edge to G, at most one base edge may have to be added to the completed graph. The problem to determine if a new base edge must be added, and if so, to find the endpoints of such a new base edge can be seen to be td-open, using a formulation of the problem in monadic second order logic. Similarly, when an edge is deleted, it may be possible that one extra edge (a base edge) disappears from the completed graph. Again, this edge can be found if necessary by solving a td-open problem, (hence in $O(\log n)$ time).

(ii) By combining techniques used in lemma 16(ii) and part (i) of this theorem. □

Corollary 18. *$(\mathcal{TW}_2, OP^-_{\mathcal{TW}_2})$ can be td-maintained with cost $O(\log n)$.*

The treewidth of the resulting tree-decompositions is at most 11. Using similar methods, one can also obtain the following result:

Theorem 19. *Let $ALMOST_k$ denote the class of almost trees with parameter k. $(ALMOST_k, OP^-_{\mathrm{ALMOST}_k})$ can be td-maintained with cost $O(\log n)$.*

6 Conclusions

By combining the results of the earlier sections, we get the following result. In all cases, directed and mixed graphs can be handeled as undirected graphs with a direction labeling on the edges.

Theorem 20. *For each problem P from table 1, there exists a data-structure that allows the following operations on graphs with treewidth ≤ 2:*

- *delete an isolated vertex*
- *add an isolated vertex*
- *delete an edge*
- *add an edge such that the treewidth of the resulting graph is at most 2*
- *check for a given pair of vertices whether adding this edge would increase the treewidth to larger than 2*
- *change the label or weight of a vertex or edge (in case P is a problem on labeled or weighted graphs)*
- *query (solve P on current graph)*

such that each operation can be carried out in $O(\log n)$ time. Queries cost $O(1)$ or $O(\log n)$ time, as described in the table. When given a graph G with treewidth ≤ 2, the data-structure can be build in $O(n)$ time.

The same result holds, when we use almost trees with parameter k for some fixed k, instead of graphs with treewidth at most 2.

An interesting open problem is to extend these results to graphs with treewidth at most k, for constant k. Using techniques from this paper (especially those in sections 3 and 4) and some older results, one can build a data-structure, that supports queries to a td-open problem P, and has the following time bounds:

- deleting an isolated vertex: $O(\log n)$ amortized, $O(n)$ worst case
- adding an isolated vertex: $O(\log n)$
- deleting an edge: $O(\log n)$
- adding an edge such that the treewidth of the resulting graph is at most k: $O(n)$ (just rebuild the data structure from scratsch ...)

Problems with $O(1)$ query time:
Vertex cover, dominating set. domatic number, chromatic number (graph coloring), achromatic number for fixed k, monochromatic triangle, feedback vertex set, feedback edge set, feedback arc set, partial feedback edge set, minimum maximal matching, partition into triangles, partition into isomorphic subgraphs for fixed H, partition into Hamiltonian subgraphs, partition into forests, partition into cliques, partition into perfect matchings for fixed k, covering by cliques, covering by complete bipartite subgraphs, clique, independent set, induced subgraph with property P (for monadic second order properties P), induced connected subgraph with property P (for monadic second order properties P), balanced complete bipartite subgraph, bipartite subgraph, degree-bounded connected subgraph for fixed k, planar subgraph, transitive subgraph, uniconnected subgraph, Hamiltonian completion, Hamiltonian path, Hamiltonian circuit, directed Hamiltonian circuit (path), subgraph isomorphism for fixed H, induced subgraph isomorphism for fixed H, path with forbidden pairs for fixed n, multiple choice matching for fixed J, kernel, k-closure, path distinguishers, degree constraint spanning tree, maximum leaf spanning tree, bounded diameter spanning tree for fixed d, kth best spanning tree for fixed k, bounded component spanning forest for fixed k, multiple choice branching for fixed m, Steiner tree in graphs, maximum cut, minimum cut into bounded sets, Chinese postman for mixed graphs, Stacker-crane, rural postman, longest circuit, chordal graph completion for fixed k, chromatic index for fixed k, spanning tree parity problem, distance d chromatic number for fixed d and k, thickness $\leq k$ for fixed k. membership for each class C of graphs that is closed under minor taking, vertex generalized geography, maximum matching, minimum spanning tree, outerplanarity, connectivity, biconnectivity, strong connectivity, triangulating colored graphs with 3 colors, (and counting variants of many of the problems described above) ...

Problems with $O(\log n)$ query time: *(endpoint of paths etc. can be specified with the query):*
longest path, shortest path, Hamiltonian path between specified endpoints, k'th shortest path for fixed k, disjoint connected paths for fixed k, maximum length-bounded disjoint paths for fixed J, maximum fixed length disjoint paths for fixed J, minimum cut between given endpoints, do s and t belong to the same: connected component, biconnected component, ... (and counting variants of many of the problems described above) ...

(See among others [2, 6].)

Table 1. List of problems

- checking for a given pair of vertices whether adding this edge would increase the treewidth to larger than k: $O(\log n)$
- changing the label or weight of a vertex or edge: $O(\log n)$
- queries: $O(1)$
- building time for given graph G with treewidth $\leq k$: $O(n)$ (with use of result in [4] and tree-contraction)

Possible improvements may well be possible here. Most challenging seems to bring down the edge insertion time. Restrictions to planar graphs, or graphs with bounded vertex degree (while still assuming an upper bound on the treewidth) seem also interesting, and may be easier to solve.

References

1. K. R. Abrahamson and M. R. Fellows. Finite automata, bounded treewidth and well-quasiordering. In *Graph Structure Theory, Contemporary Mathematics vol. 147*, pages 539–564. American Mathematical Society, 1993.
2. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12:308–340, 1991.
3. H. L. Bodlaender. NC-algorithms for graphs with small treewidth. In J. van Leeuwen, editor, *Proc. Workshop on Graph-Theoretic Concepts in Computer Science WG'88*, pages 1–10. Springer Verlag, Lecture Notes in Computer Science, vol. 344, 1988.
4. H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the 25th Annual Symposium on Theory of Computing*, pages 226–234. ACM Press, 1993.
5. H. L. Bodlaender and T. Kloks. A simple linear time algorithm for triangulating three-colored graphs. *J. Algorithms*, 15:160–172, 1993.
6. R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–582, 1992.
7. N. Chandrasekharan. *Fast Parallel Algorithms and Enumeration Techniques for Partial k-Trees.* PhD thesis, Clemson University, 1990.
8. R. F. Cohen, S. Sairam, R. Tamassia, and J. S. Vitter. Dynamic algorithms for bounded tree-width graphs. Technical Report CS-92-19, Department of Computer Science, Brown University, 1992.
9. D. Fernández-Baca and G. Slutzki. Parametic problems on graphs of bounded treewidth. In O. Nurmi and E. Ukkonen, editors, *Proceedings 3rd Scandinavian Workshop on Algorithm Theory*, pages 304–316. Springer Verlag, Lecture Notes in Computer Science, vol. 621, 1992.
10. G. N. Frederickson. A data structure for dynamically maintaining rooted trees. In *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms*, pages 175–184, 1993.
11. G. N. Frederickson. Maintaining regular properties dynamically in k-terminal graphs. Manuscript, 1993.
12. J. Lagergren. Efficient parallel algorithms for tree-decomposition and related problems. In *Proceedings of the 31rd Annual Symposium on Foundations of Computer Science*, pages 173–182, 1990.
13. J. Matousěk and R. Thomas. Algorithms finding tree-decompositions of graphs. *J. Algorithms*, 12:1–22, 1991.
14. G. L. Miller and J. Reif. Parallel tree contraction and its application. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, pages 478–489, 1985.
15. B. Reed. Finding approximate separators and computing tree-width quickly. In *Proceedings of the 24th Annual Symposium on Theory of Computing*, pages 221–228, 1992.
16. N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.
17. J. van Leeuwen. Graph algorithms. In *Handbook of Theoretical Computer Science, A: Algorithms and Complexity Theory*, pages 527–631, Amsterdam, 1990. North Holland Publ. Comp.