Improved bottleneck domination algorithms

Ton Kloks[1], Dieter Kratsch[2*], C. M. Lee[3], and Jiping Liu[1**]

[1] Department of Mathematics and Computer Science
The University of Lethbridge
Alberta, T1K 3M4, Canada
(kloks|liu)@cs.uleth.ca
[2] Université de Metz
LITA
57045 Metz, Cedex 01, France
kratsch@sciences.univ-metz.fr
[3] Department of Computer Science and Information Engineering
National Chung Cheng University
Chiayi, Taiwan
cmlee@cs.ccu.edu.tw

Abstract. W.C. Yen introduced BOTTLENECK DOMINATION and BOTTLENECK INDEPENDENT DOMINATION. He presented an $O(n \log n + m)$ algorithm to compute a minimum bottleneck dominating set. He also showed that the BOTTLENECK INDEPENDENT DOMINATING SET problem is NP-complete, even when restricted to planar graphs. He also obtained a linear time algorithm to solve this problem for interval graphs.
We present linear time algorithms for the BOTTLENECK DOMINATING SET and the BOTTLENECK TOTAL DOMINATING SET problem. Furthermore we give polynomial time algorithms (most of them with linear running time) for the BOTTLENECK INDEPENDENT DOMINATING SET problem on the following graph classes: AT-free graphs, chordal graphs, split graphs, permutation graphs, graphs of bounded treewidth, and graphs of bounded clique-width.

1 Introduction

We consider undirected, finite and simple graphs $G = (V, E)$. We denote by $G[W]$ the subgraph of $G = (V, E)$ induced by the vertex set $W \subseteq V$. For a vertex $x \in V$ in a graph $G = (V, E)$ we use $N(x)$ to denote its neighborhood and $N[x]$ to denote its closed neighborhood $N[x] = N(x) \cup \{x\}$. For $S \subseteq V$, let $N(S) = \bigcup_{s \in S} N(s)$ and $N[S] = \bigcup_{s \in S} N[s]$.
Let $w : V \to \Re$ be a function assigning to each vertex v of G a weight $w(v)$ such that all arithmetic operations on vertex weights can be performed in time $O(1)$. We call w a vertex weight function and (G, w) a weighted graph.
We were motivated by the work of C.W. Yen who introduced the problems BOTTLENECK DOMINATING SET and BOTTLENECK INDEPENDENT DOMINATING

* Corresponding author.
** Partially supported by NSERC of Canada.

SET in [17]. He showed that the bottleneck version of the NP-complete graph problem (MINIMUM WEIGHT) DOMINATING SET is solvable by an $O(m + n \log n)$ algorithm. On the other hand, the bottleneck version of the NP-complete problem (MINIMUM WEIGHT) INDEPENDENT DOMINATING SET remains NP-complete (even on planar graphs).

Definition 1. *The* BOTTLENECK (TOTAL) (INDEPENDENT) DOMINATING SET problem *BDS (BTDS) (BIDS) asks for a (total) (independent) dominating set D of a weighted graph (G, w) such that the maximum weight over the vertices in D is as small as possible.*

In a similar way the problems BOTTLENECK CONNECTED DOMINATING SET (where $G[D]$ is to be connected) and BOTTLENECK DOMINATING CLIQUE (where D is to be a clique) can be defined.
In our paper we present the following results. In Section 2 we show that there is linear time algorithms to compute a minimum bottleneck dominating set. In a similar way a linear time algorithm to compute a minimum bottleneck total dominating set can be obtained. Similar results can be obtained for bottleneck k-tuple domination, bottleneck distance two domination, etc.
The remainder of the paper is dedicated to the BOTTLENECK INDEPENDENT DOMINATING SET problem. In section 3 we study the general relations between bottleneck, constrained and minimum weight independent domination. In section 4 we present efficient algorithms for the NP-complete problem bottleneck independent dominating set on the following graph classes: AT-free graphs, chordal graphs, split graphs, permutation graphs, graphs of bounded treewidth, and graphs of bounded clique-width for which a k-expression describing the graph is part of the input.
For the definition of graph classes and their properties not given in this extended abstract we refer to [3].

2 Linear time algorithms for BDS and BTDS

The decision variants of the bottleneck problems have a real constant η as part of the input, and the question is asked whether there exists a bottleneck (independent/total) dominating set with maximum weight (or *bottleneck*) at most η.
Given any constant η it is easy to check whether there exists a dominating, total dominating, connected dominating set D, respectively, with maximum weight at most η: simply take $D_\eta = \{x \mid w(x) \leq \eta\}$. Note that any superset of a dominating, total dominating, connected dominating set, respectively, is also a dominating, total dominating, connected dominating set, respectively. Hence if D_η is dominating, total dominating, connected dominating, respectively, then there exists a dominating, total dominating, connected dominating set of bottleneck at most η, respectively. Otherwise not. In [17] this observation is used to devise a binary search based algorithm that runs in time $O(n \log n + m)$ to solve the BOTTLENECK DOMINATING SET problem.

In this section we show that the BOTTLENECK DOMINATING SET problem can be solved in linear time for all weighted graphs (G, w).

Definition 2. $\forall_{x \in V}: m(x) = \min\{w(u) \mid u \in N[x]\}$. *Let* $\beta = \max_{x \in V} m(x)$.

Theorem 1. *The minimum bottleneck of a dominating set in a weighted graph* (G, w) *is* β.

Proof. Consider a dominating set D of G with bottleneck η. Then, since for every vertex x of G its closed neighborhood $N[x]$ has a vertex in D, $m(x) \leq \eta$ for all vertices x of G. Hence $\beta \leq \eta$.
On the other hand, there exists a dominating set with bottleneck β. This can be seen by choosing any one vertex y from every closed neighborhood $N[x]$ with $w(y) \leq \beta$ in D (such a vertex must exist). Since every closed neighborhood has a vertex in D, this set must be dominating. □

Clearly, β can be computed in $O(n + m)$ time, and thus:

Corollary 1. *The* BOTTLENECK DOMINATING SET *problem can be solved in* $O(n + m)$ *time for any weighted graph* (G, w).

In a similar way, i.e. we only have to redefine $m(x)$, one can obtain a $O(n + m)$ time algorithm to compute a minimum bottleneck total dominating set for any weighted graph (G, w).
A double dominating set of a graph $G = (V, E)$ is a set D such that for all $x \in V$, $|N[x] \cap D| \geq 2$. For the BOTTLENECK DOUBLE DOMINATING SET problem we can also obtain a linear time algorithm.

3 Bottleneck, constrained and minimum weight IDS

The following two problems are closely related to the BOTTLENECK INDEPENDENT SET PROBLEM.

Definition 3. *The* CONSTRAINED INDEPENDENT DOMINATING SET *problem (CIDS) is the following decision problem:*
INSTANCE: *A graph* $G = (V, E)$ *and a set* $V' \subseteq V$.
QUESTION: *Does there exist an independent set* $S \subseteq V - V'$ *which dominates* V'*, i.e. every vertex of* V' *has a neighbour in* S*?*

An examination of the proof of Lemma 4 in [17] shows that Yen established:

Lemma 1. *The problems* CONSTRAINED INDEPENDENT DOMINATING SET *and the decision version of* BOTTLENECK INDEPENDENT DOMINATING SET *are reducible to each other in time* $O(n)$.

His reductions go as follows: Given (G, w, η) as input for the decision version of BIDS, take (G, V') as input for CIDS where $v \in V'$ iff $w(v) > \eta$. Given (G, V') as input for CIDS, take (G, w, η) as input for the decision version of BIDS where $\eta = 1.5$, and for all $v \in V$, $w(v) = 2$ if $v \in V'$ and $w(v) = 1$ otherwise.

Definition 4. *The* MINIMUM WEIGHT INDEPENDENT DOMINATING SET *problem (WIDS) is the following problem:*
INSTANCE: *A weighted graph* (G, w).
QUESTION: *Find the minimum weight of an independent dominating D of G, i.e. minimize* $\sum_{v \in D} w(v)$ *over all independent dominating sets D of G.*

Lemma 2. *The decision version of the* BOTTLENECK INDEPENDENT DOMINATING SET *problem can be reduced in time $O(n)$ to the decision version of the* MINIMUM WEIGHT INDEPENDENT DOMINATING SET *problem. Moreover, the decision version of BIDS and the decision version of the* (0-1) INDEPENDENT DOMINATING SET *problem asking whether a graph G with $0-1$ vertex weight function w has an independent dominating set of weight* 0 *are reducible to each other in time $O(n)$.*

Proof. Consider an instance (G, w, η) of the decision problem BIDS. Construct an instance $(G, w', 0)$ of the decision version of WIDS as follows. To each vertex v of G assign weight $w'(v) = 0$ if $w(v) \leq \eta$ and $w'(v) = 1$ otherwise. Then there is an independent dominating set of G with bottleneck at most η if and only if (G, w') has an independent dominating set of weight 0. $\square$

Similarly one can show.

Lemma 3. *The problem CIDS can be reduced in time $O(n)$ to the decision version of WIDS and to (0-1)IDS.*

Consequently using binary search on the weights of the vertices (and possibly some preprocessing sorting step) BIDS can be solved using the decision version of WIDS or (0-1)IDS with an overhead of a $\log n$ factor. BIDS can be solved using CIDS without increase of the bound for the running time.
Finally let us emphasize that none of the reductions of this section changes the graph, thus all of the above lemmas remain true when restricted to any graph class. Summarizing we obtain.

Theorem 2. *Let $\mathcal{G}$ be any class of graphs. Let $t_B(n,m)$, $t_C(n,m)$, $t_{01}(n,m)$ and $t_W(n,m)$ be the best worst-case running time of an algorithm for BIDS, CIDS, the decision version of (0-1)IDS and the decision version of WIDS on graph class $\mathcal{G}$. Then*

(i) $t_B(n,m) = O(\log n \ t_C(n,m))$,
(ii) $t_B(n,m) = O(\log n \ t_{01}(n,m))$,
(iii) $t_B(n,m) = O(\log n \ t_W(n,m))$,
(iv) $t_{01}(n,m) = O(t_W(n,m))$,
(v) $t_C(n,m) = O(t_B(n,m))$,
(vi) $t_C(n,m) = O(t_{01}(n,m))$,
(vii) $t_C(n,m) = O(t_W(n,m))$.

Consequently if WIDS, (0-1)IDS or CIDS is polynomial time solvable on some graph class $\mathcal{G}$, then BIDS is also polynomial time solvable on $\mathcal{G}$. However it remains possible that WIDS is NP-complete on a graph class $\mathcal{G}'$ while BIDS is polynomial time solvable on $\mathcal{G}'$.

Remark 1. The problem CIDS is also introduced in [12] were it is called RED MAXIMAL INDEPENDENT SET. It was shown that CIDS can be solved in polynomial time for circle graphs. This is very interesting since it shows that BIDS is polynomial time solvable on circle graphs while even INDEPENDENT DOMINATING SET remains NP-complete on circle graphs [7]. Thus the complexities of BIDS and INDEPENDENT DOMINATING SET differ on circle graphs.

4 Bottleneck independent dominating set

In this section we present algorithms to compute a minimum bottleneck independent dominating set on the following graph classes: AT-free graphs, chordal graphs, split graphs, permutation graphs, graphs of bounded treewidth and graphs of bounded clique-width (when a so-called k-expression describing the graph is part of the input). Most of our algorithms have linear running time.

4.1 Permutation graphs

It is mentioned in [16] that the WIDS problem on permutation graphs can be solved in linear time. (See also [14].) Hence BIDS on permutation graphs is $O((n+m)\log n)$ time solvable on permutation graphs by Theorem 2. In this subsection we show that there is a linear time algorithm to compute a minimum bottleneck independent dominating set for permutation graphs (i.e. we show how to avoid the binary search and get rid of the $\log n$ factor in the running time).
We find it convenient to view upon permutation graphs as intersection graphs as follows. Write the number 1 up to n horizontally from left to right. Underneath write the numbers $\pi(1),\ldots,\pi(n)$ also from left to right. Then connect the two 1's, the two 2's etc. by straight line segments. Then two vertices x and y are adjacent whenever the corresponding line segments intersect. This intersection model is called a *permutation diagram*. We label the line segment joining the two i's with label i and in the following we identify the line and its corresponding vertex in the graph $G(\pi)$.
Notice that given a permutation diagram of the graph $G=(V,E)$ and a vertex weight function w, then for a line x we could define an immediate successor as a line y such that y lies to the right of x and there is no other parallel line between x and y, i.e. there is no line $z \notin \{x,y\}$ between x and y that crosses neither x nor y. Notice that the direct successors of a line must form a clique in the graph $\tilde{G}$. We shall describe a dynamic programming algorithm using these cliques.

Definition 5. *Any two lines $i,j \in V_n$ with $i \leq j$ are a* crossing pair*, denoted by $\mathbb{X}_{ij}$, if $i=j$ or i and j intersect.*

For each crossing pair $\mathbb{X}_{ij}$ define $V_{ij} = \{k \mid k \leq j \wedge \pi^{-1}(k) \leq \pi^{-1}(i)\}$. and $V'_{ij} = \{k \mid k < i \wedge \pi^{-1}(k) < \pi^{-1}(j)\}$. Hence V'_{ij} consists of those lines which lie completely to the left of the lines i and j.
We denote by D_{ij} any independent dominating set of smallest possible maximum weight of $G[V_{ij}]$. Consequently $D_{\pi(n)n}$ is a minimum bottleneck independent dominating set of G.

Definition 6. *For a crossing pair $\mathbb{X}_{ij}$, if $V'_{ij} = \emptyset$ then define $\beta(\mathbb{X}_{ij}) = \emptyset$. Otherwise let $\beta(\mathbb{X}_{ij})$ be the crossing pair $\mathbb{X}_{i_b j_b}$ satisfying $i_b, j_b \in V'_{ij}$, j_b is the largest number in V'_{ij}, and $\pi^{-1}(i_b) \geq \pi^{-1}(k)$ for all $k \in V'_{ij}$.*

Hence $\mathbb{X}_{i_b j_b}$ is the crossing pair having j_b as the rightmost top endpoint among all lines in V'_{ij} and having i_b as the rightmost bottom endpoint among all lines in V'_{ij}.

Lemma 4. *Let $j \in V_n$. If $V'_{jj} \neq \emptyset$ then $\beta(\mathbb{X}_{jj}) = \mathbb{X}_{i_b j_b}$ and $D_{jj} = D_{i_b j_b} \cup \{j\}$. If $V'_{jj} = \emptyset$ then $\beta(\mathbb{X}_{jj}) = \emptyset$ and $D_{jj} = \{j\}$.*

Proof. The vertex j is isolated in $G[V_{jj}]$. Hence any dominating set must contain j. Thus, if $\beta(\mathbb{X}_{jj}) \neq \emptyset$, then $V_{j_b j_b} \cup \{j\} = V_{jj}$. Therefore D is an independent dominating set of $G[V_{jj}]$ if and only if $j \in D$ and $D - \{j\}$ is an independent dominating set of $G[V_{j_b j_b}]$. ☐

Definition 7. *For each crossing pair $\mathbb{X}_{ij}$ with $i < j$ define the clique $\mathcal{C}(\mathbb{X}_{ij})$ as the set of all lines k with $i \leq k \leq j$ and $\pi^{-1}(j) \leq \pi^{-1}(k) \leq \pi^{-1}(i)$, and such that there is no line ℓ with $k < \ell < j$ and $\pi^{-1}(k) < \pi^{-1}(\ell) < \pi^{-1}(i)$.*

Lemma 5. *Let $\mathbb{X}_{ij}$ be a crossing pair with $i < j$. Then there is a line $k \in \mathcal{C}(\mathbb{X}_{ij})$ such that D_{kk} is an independent dominating set of minimum bottleneck in $G[V_{ij}]$.*

Proof. Let D be an independent dominating set of $G[V_{ij}]$ of smallest maximum weight. Clearly $|D \cap \mathcal{C}(\mathbb{X}_{ij})| \leq 1$. Let k be the rightmost line in D. Since D is a dominating set, there cannot be any line ℓ completely to the right of k. Hence $k \in \mathcal{C}(\mathbb{X}_{ij})$. By Lemma 4, D_{kk} is an independent dominating set in $G[V_{ij}]$ of smallest maximum weight and D_{kk} has rightmost line k for some $k \in \mathcal{C}(\mathbb{X}_{ij})$. Hence the bottleneck of D is equal to the bottleneck of D_{kk}. ☐

Definition 8. *Define $\delta(\mathbb{X}_{ii}) = i$, and for $i < j$, define $\delta(\mathbb{X}_{ij})$ to be the largest number in $\mathcal{C}(\mathbb{X}_{ij}) - \{j\}$.*

Hence $\delta(\mathbb{X}_{ij})$ is the line in $\mathcal{C}(\mathbb{X}_{ij})$ with rightmost top endpoint, and its top endpoint is closest to j among all lines of $\mathcal{C}(\mathbb{X}_{ij})$.

Lemma 6. *Let $i < j$ and $t = \delta(\mathbb{X}_{ij})$. Then the smallest maximum weight of an independent dominating set of $G[V_{ij}]$ is equal to the minimum of the maximum weight of D_{jj} and the maximum weight of D_{it}.*

Proof. By Lemma 5, there exists a $k \in \mathcal{C}(\mathbb{X}_{ij})$ such that D_{kk} is a minimum bids of $G[V_{ij}]$. If $k = j$ then D_{jj} is a minimum bids. Otherwise $k < j$ and then $k \in \mathcal{C}(\mathbb{X}_{it}) = \mathcal{C}(\mathbb{X}_{ij}) - \{j\}$ and D_{kk} is an independent dominating set of minimum bottleneck. Consequently D_{it} is a minimum bids of $G[V_{ij}]$. ☐

$\delta(\mathbb{X}_{ij})$ and $\beta(\mathbb{X}_{ij})$ can be computed in linear time for all crossing pairs $\mathbb{X}_{ij}$ (see [16]).

Theorem 3. *There is a linear time algorithm to compute a minimum bottleneck independent dominating set for permutation graphs.*

Proof. Define an ordering $X_{ij} < X_{k\ell}$ if and only if $j < \ell$ or $j = \ell$ and $i > k$. Use the formulas of Lemma 4 and Lemma 6 to compute the minimum bottleneck of an independent dominating set in $G[V_{ij}]$ in increasing order of the crossing pairs $\mathbb{X}_{ij}$ using dynamic programming. Finally, $D_{\pi(n)n}$ is a minimum bottleneck independent dominating set of the graph $G \cong G(\pi)$. $\square$

4.2 Split graphs, chordal, and AT-free graphs

A split graph $G = (V, E)$ is a graph for which the vertex set V can be partitioned into an independent set S and a clique C.

Theorem 4. *The* BOTTLENECK INDEPENDENT DOMINATING SET *problem can be solved in $O(n \log n + m)$ time for split graphs.*

Proof. First the algorithm sorts the vertices of the independent set S according to non-decreasing weights. Say $S = \{s_1, \ldots, s_\ell\}$ with $w(s_1) \geq \ldots \geq w(s_\ell)$.

Now, for each vertex $r \in C$, the algorithm determines the minimum index $j(r)$ such that $s_{j(r)} \notin N(r)$ in the following way: for each vertex $r \in C$ the algorithm initializes $j(r) = 1$. Then for $i := 1$ to ℓ it passes through the adjacency list of s_i and for all neighbours $r \in C$ of s_i with $j(r) = i$ it increments $j(r)$.

Let $m(r) = \max\{w(r), w(s_{j(r)})\}$. Clearly, computing $m(r)$ for all $r \in C$ takes $O(n + m)$ time. Finally, compute $\eta = \min_{r \in C} m(r)$.

If $\eta \leq w(s_1)$ then η is the minimum bottleneck of an independent dominating set. Otherwise the minimum bottleneck is $w(s_1)$. $\square$

Chordal graphs form a superclass of split graphs. In [8] M. Farber presents a linear algorithm to locate a minimum weight independent dominating set in a chordal graph with 0-1 vertex weights based on a linear programming formulation of the problem. Using Theorem 2 we obtain immediately:

Theorem 5. *There is a $O((n + m) \log n)$ algorithm to compute a minimum bottleneck independent set on chordal graphs.*

In [8] M. Farber also mentions a private communication by G.J. Chang saying that WIDS on chordal graphs is NP-complete. Thus chordal graphs are another graph class for which the complexity of BIDS and WIDS differs. In fact the difference in complexity is based on a difference in complexity of WIDS and (0-1)IDS first discovered by M. Farber.

There is a $O(n^4)$ algorithm for WIDS of AT-free graphs in [4], and thus BIDS can be solved in time $O(n^4 \log n)$ on AT-free graphs. Upon close examination of the method used in [4] it is clear that the BIDS problem can be solved directly in $O(n^4)$ time for AT-free graphs: in the formulas in [4] giving the weighted independent dominating set summation has to be replaced by maximization.

Theorem 6. *The* BOTTLENECK INDEPENDENT DOMINATING SET *problem can be solved in time $O(n^4)$ on AT-free graphs.*

4.3 Graphs of bounded treewidth

Let (G, w) be a weighted graph and assume that G has treewidth at most k (for some constant k). We present an explicit $O(5^k n)$ time algorithm making use of a *nice tree decomposition*. Such a nice tree decomposition (with a linear number of nodes) can be obtained in linear time [2, 10].

A tree decomposition $(T, \mathcal{S})$ is *rooted* if the tree T is equipped with some root node. A rooted tree decomposition is called *nice* if every node of T has at most two children and the following conditions are satisfied. If a node i has two children j and k then $S_i = S_j = S_k$ and in this case we call the node i a *join node*. Also, if a node i has only one child j then either $|S_i| = |S_j| + 1$ and $S_j \subset S_i$ (in which case the node i is called an *introduce node*) or $|S_i| = |S_j| - 1$ and $S_i \subset S_j$ (and in this case the node i is called a *forget node*). Finally, if a node i has no children and is not the root, it is called a *start node*.

For a node i in the tree consider the subtree T_i of T rooted at i. We let G_i stand for the subgraph of G induced by vertices in bags $S_j \in \mathcal{S}$ with j a node of T_i. Our algorithm works from the leaves in T up to the root, computing partial solutions for G_i on the way. These partial solutions are characterized by the status of the nodes in S_i. The status of a node $x \in S_i$ can be of three different types indicated by one of the colors black, grey, and white.

A partial solution for $G_i = (V_i, E_i)$ consists of an independent set $D_i \subseteq V_i$ such that all vertices of $V_i - S_i$ are dominated by vertices of D_i. Furthermore, vertices in $D_i \cap S_i$ are *black*. Vertices of $S_i - D_i$ which are dominated by D_i are colored *grey*. And vertices of $S_i - D_i$ which are not dominated by D_i are *white*.

For a node i in the tree we keep a table of all colorings of the vertices of S_i for which there exists some partial solution D_i for G_i. Furthermore, for each such coloring we also store the minimum maximal weight of a partial solution D_i giving rise to this coloring.

We now describe how to obtain a table of partial solutions for a node i from the tables of its children. We consider the four different cases.

Node i is a start node. In this case all colorings of vertices in S_i are stored if the black vertices are independent and the grey vertices are exactly those which have a black neighbour. For each such coloring the maximum weight of the black vertices is also stored. If there are no black vertices, this entry is given a value ∞.

Node i is a forget node. Let $S_i = S_j - x$. Consider the different colorings in the table stored at node j. If the color of x is *white*, then this can no longer be a partial solution since for partial solutions it is required that all vertices in $V_i - S_i$ are dominated. In all other case, the partial solution is stored as a partial solution in the table at node i, of course without the vertex x. The table at node i is reduced so that it contains no multiple equal entries, i.e., equal except for the value stored. Of course, for the value of such 'equal' entries the minimum is taken.

Node i is a join node. Let j and h be the two children of i in T. Consider the partial solutions for G_j and G_h characterized by the tables at these nodes.

Since $S_i = S_j = S_h$ a coloring for S_i can correspond to a partial solution only if the two sets of black vertices in S_j and S_h are exactly the same. For such corresponding entries at j and h the table at node i contains the *union* of the grey vertices (since a vertex is dominated in G_i if it is dominated in at least one of G_j and G_h). A vertex is white if and only if it is white both in the table entry for j and h. The *value* of this partial solution stored at node i is the maximum of the two values stored at j and h. Again, the table (at node i) is reduced such that it contains no entries which are equal up to the value, and for such 'equal' entries the minimum value is taken.

Node i is an introduce node. Let $S_i = S_j + x$. Consider again the partial solutions for G_j characterized by the colorings in the table for node j. If x has a black neighbour, then clearly it can only be colored grey. Assume that x has no black neighbour. In that case it is not dominated by the partial solution since all neighbours of x in G_i must be vertices in S_i. We now have two possibilities for creating partial solutions at node i.
The first option is that we color x white. In that case the smallest maximum weight of a partial solution doesn't change, and neither the color of any vertex in S_i changes.
Secondly, we may choose to color the vertex x black. First of all, we now may have to update the smallest maximal weight of a partial solution with this characteristic if the weight of x exceeds this value. But also, vertices in S_i which were colored white and which are neighbours of x now are colored grey.

Theorem 7. *For each constant $k \geq 1$, there is an $O(5^k n)$ time algorithm to solve the BIDS problem for graphs of treewidth at most k.*

Proof. Consider a join node i with children j and h. For one grey vertex in the table entry for i, one has to compute the minimum value of three entries in j and h: grey and white, white and grey, and grey and grey. Let s be the size of S_i, then there are at most $\sum_{m=0}^{s} \binom{s}{m} 2^{s-m} 3^m = 5^s$ computations to make for the table at node i. This leads to the overall running time of $O(5^k n)$ for this algorithm (see also [1]).
Consider the partial solutions at the root node. Choose the table entry without white vertex with the smallest value attached to it. This corresponds to an independent dominating set of minimum maximal weight.
If we keep pointers at the table entries, telling where the solutions came from we can also find a minimum bids of G within the same timebound. ◻

4.4 Graphs of bounded clique-width

In this section we show that for graphs of clique-width at most k and with a given k-expression, the BIDS problem can be solved in $O(5^k k^3 n)$ time. The method we use is similar to the one used in [13] to obtain an $O(2^{4k} k^2 n)$ time algorithm for the DOMINATING SET problem.
Graphs of clique-width at most k can be recursively defined by the following four operations on graphs with vertex labels from $\{1, \ldots, k\}$. The clique-width

of a graph is the smallest k for which the graph can be constructed by these four operations such that only labels from $\{1, \ldots, k\}$ are used. A *k-expression* is a description of the construction of the graph via these four operations.

- $\mathbf{i(v)}$: The operation creates a new vertex v with label i (incident with no edges).
- $\mathbf{G \oplus H}$: The operation takes as input two labelled graphs and creates their disjoint union.
- $\eta_{\mathbf{i,j}}$: The operation adds edges between all vertices with label i and all vertices of label j.
- $\rho_{\mathbf{i \to j}}$: The operation relabels all vertices with label i to label j.

Notice that there always exists a k-expression with at most $O(k^2 n)$ operations (or a corresponding decomposition tree with at most $O(k^2 n)$ nodes). This follows since there is no operation *removing* any vertex from a graph.

Definition 9. *Let S_1, S_2, and S_3 be three* disjoint *subsets of $\{1, \ldots, k\}$. We write $V(S_1)$ for the set of vertices of V with a label in S_1. Define $\Gamma(S_1, S_2, S_3)$ as the smallest maximum weight of an independent dominating set of $G[V - V(S_1)]$ which contains no vertex with a label in S_2, and which contains for each label in S_3 at least one vertex with this label. If $V - V(S_1) = \emptyset$ or if there are no vertices for some label in S_3, we define $\Gamma(S_1, S_2, S_3) = 0$, otherwise, if such an independent dominating set does not exist we define $\Gamma(S_1, S_2, S_3) = \infty$.*

Notice that the minimum bottleneck of an independent dominating set of G is equal to $\Gamma(\emptyset, \emptyset, \emptyset)$.
Now we show how the values $\Gamma(S_1, S_2, S_3)$ can be computed for each of the four operations. To avoid confusion we write $\Gamma(S_1, S_2, S_3)$ for the values after the operation and $\gamma(S_1, S_2, S_3)$ the values before the operation. Due to space limitations we have to omit the correctness proofs of the formulas.

$\mathbf{i(v)}$:

$$\Gamma(S_1, S_2, S_3) = \begin{cases} \infty & i \in S_2 \\ \gamma(S_1, S_2, S_3) & i \in S_1 \\ \max(\gamma(S_1, S_2, S_3), w(v)) & i \notin S_1 \cup S_2 \end{cases}$$

$\mathbf{G \oplus H}$: $\Gamma(S_1, S_2, S_3) = \min\limits_{S', S'' \subseteq S_3} (\max(\gamma_G(S_1, S_2, S'), \gamma_H(S_1, S_2, S'')) ,$
where the minimum is taken over all partitions of S_3 into two subsets S' and $S'' = S_3 - S'$.

$\eta_{\mathbf{i,j}}$: **Of course in this case we cannot have $\mathbf{i \in S_3}$ and $\mathbf{j \in S_3}$.**

$$
\Gamma(S_1,S_2,S_3) = \begin{cases} \gamma(S_1+j, S_2-j, S_3) & i \in S_3 \wedge j \notin S_1 \\ \gamma(S_1+i, S_2-i, S_3) & j \in S_3 \wedge i \notin S_1 \\ \min(\gamma(S_1+i, S_2-i, S_3+j), & \\ \qquad \gamma(S_1, S_2+j, S_3)) & i \in S_2 \wedge j \notin S_1 \cup S_2 \cup S_3 \\ \min(\gamma(S_1+j, S_2-j, S_3+i), & \\ \qquad \gamma(S_1, S_2+i, S_3)) & i \notin S_1 \cup S_2 \cup S_3 \wedge j \in S_2 \\ \min(\gamma(S_1+i, S_2, S_3+j), & \\ \qquad \gamma(S_1+j, S_2, S_3+i), & \\ \qquad \gamma(S_1, S_2+i+j, S_3)) & i,j \notin S_1 \cup S_2 \cup S_3 \\ \gamma(S_1, S_2, S_3) & i \in S_1 \vee j \in S_1 \vee i,j \in S_2 \end{cases}
$$

$\rho_{\mathbf{i}\to\mathbf{j}}$**: Notice that we may assume that** $\mathbf{i} \notin \mathbf{S_1} \cup \mathbf{S_2} \cup \mathbf{S_3}$.

$$
\Gamma(S_1,S_2,S_3) = \begin{cases} \gamma(S_1+i, S_2, S_3) & j \in S_1 \\ \gamma(S_1, S_2+i, S_3) & j \in S_2 \\ \gamma(S_1, S_2, S_3+i) & j \in S_3 \\ \gamma(S_1, S_2, S_3) & j \notin S_1 \cup S_2 \cup S_3 \end{cases}
$$

Theorem 8. *For each $k \geq 1$, there is an $O(5^k k^3 n)$ time algorithm to compute a minimum bottleneck independent dominating set of graphs of clique-width at most k assuming the graph is given with a k-expression.*

Proof. The subsets S_1, S_2 and S_3 are disjoint. For the computation of the disjoint union we have to try all $O(2^{|S_3|})$ possible partitions of the set S_3. Notice that

$$
\sum_{0 \leq |S_1|+|S_2|+|S_3| \leq k} \binom{k}{|S_1|} \binom{k-|S_1|}{|S_2|} \binom{k-|S_1|-|S_2|}{|S_3|} 2^{|S_3|} = 5^k
$$

There are $O(k^2 n)$ nodes in the decomposition tree. To obtain explicit list of the sets S_i, we need $O(k)$ time per node in the decomposition tree. Finally an independent dominating set of minimum bottleneck can be found using pointers in the standard way. ☐

For some graph classes such as cographs, forests and distance hereditary graphs $O(n+m)$ algorithms to compute a k-expression (here $k=2$ or $k=3$) are known.

Corollary 2. *There is an $O(n+m)$ time algorithm to compute a minimum bottleneck independent dominating set for distance hereditary graphs.*

5 Conclusions

WIDS on cocomparability graphs can be solved by an $O(n^3)$ dynamic programming algorithm [15]. Breu and Kirkpatrick mention an $O(n^{2.376})$ algorithm that seemingly they never got published. The $O(n^3)$ algorithm can easily be turned into an $O(n^3)$ algorithm for BIDS since we only have to change the objective function for the dynamic programming. Is there a faster algorithm to solve BIDS on cocomparability graphs such as one with running $O(n^2)$ or even $O(n+m)$? The algorithmic complexities of WIDS and BIDS differ for circle graphs and for chordal graphs. It would be nice to find more graph classes for which WIDS or even (0-1)IDS are NP-complete while BIDS is polynomial.

References

1. Alber, J., H. L. Bodlaender, H. Fernau, T. Kloks, R. Niedermeier, Fixed parameter algorithms for dominating set and related problems on planar graphs, *Algorithmica* **33** (2002), 461–493.
2. Bodlaender, H., A linear time algorithm for finding tree–decompositions of small treewidth, *SIAM J. Comput.* **25** (1996), 1305–1317.
3. Brandstädt, A., V. Bang Le, and J. Spinrad, *Graph classes–A survey*, SIAM Monographs on Discrete Mathematics and Applications, Philadelphia 1999.
4. Broersma, H., T. Kloks, D. Kratsch, and H. Müller, Independent sets in asteroidal triple–free graphs, *SIAM J. Discrete Math.* **12** (1999), 276–287.
5. Corneil, D. G., M. Habib, J. Lanlignel, B. Reed, and U. Rotics, Polynomial time recognition of clique-width ≤ 3 graphs (ext. abstract), *Proceedings of LATIN 2000*, Lecture Notes in Comput. Sci. **1776**, Springer, 2000, pp 126–134.
6. Courcelle, B., Graph rewriting: an algebraic and logical approach, In (Jan van Leeuwen ed.) *Handbook of Theoretical Computer Science, Vol. B*, Amsterdam, Elsevier Science Publ., 1990, pp. 192–242.
7. Damian-Iordache, M. and S. Pemmaraju, Hardness of approximating independent domination in circle graphs, *Proceedings of ISAAC 1999*, Lecture Notes in Comp. Sci. **1741**, Springer, 1999, pp. 56–69,
8. Farber, M., Independent domination in chordal graphs, *Oper. Res. Lett.* **1** (1981/82), 134–138.
9. Golumbic, M. C. and U. Rotics, On the clique–width of some perfect graph classes, *Internat. J. Found. Comp. Sci.* **11** (2000), 423–443.
10. Kloks, T., *Treewidth–Computations and Approximations*, Lecture Notes Comp. Sci. **842**, Springer, 1994.
11. Kloks, T., C. M. Lee, and J. Liu, New algorithms for k-Face Cover, k-Feedback Vertex Set and k-Disjoint Cycles in plane and planar graphs, *Proceedings of WG 2002*, Lecture Notes Comp. Sci. **2573**, Springer, 2002, pp. 282–295.
12. Kloks, T., C. M. Lee, J. Liu, and H. Müller, Blanketing general partition graphs, to appear in *Proceedings of WG 2003*.
13. Kobler, D. and U. Rotics, Polynomial algorithms for partitioning problems on graphs with fixed clique–width (extended abstract), *Proceedings of the 12th Annual ACM–SIAM Symposium on Discrete Algorithms*, 2001, pp. 468–476.
14. Kratsch, D., *Algorithms*, Chapter 8 in: *Domination in Graphs: Advanced Topics*, T. Haynes, S. Hedetniemi, P. Slater, (eds.), Marcel Dekker, 1998, pp. 191–231.
15. Kratsch, D. and L. Stewart, Domination on cocomparability graphs, *SIAM J. Discrete Math.* **6**, (1993), 400–417.
16. Liang, Y. D. and C. Rhee, Linear algorithms for two independent set problems on permutation graphs, *Proc. of the 22nd Comp. Sci. Conference*, (1994), pp. 90–93.
17. Yen, W. C., Bottleneck domination and bottleneck independent domination on graphs, *Journal of Information Science and Engineering* **18** (2002), pp. 311–331.