Theoretical Computer Science 410 (2009) 5455–5466

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Computational study on planar dominating set problem☆

Marjan Marzban [a], Qian-Ping Gu [a,*], Xiaohua Jia [b]

[a] School of Computing Science, Simon Fraser University, Burnaby, BC, Canada, V5A 1S6

[b] Department of Computer Science, City University of Hong Kong, Hong Kong

ARTICLE INFO

Keywords:
PLANAR DOMINATING SET
Branch-decomposition
Fixed-parameter algorithms
Data reduction
Computational study

ABSTRACT

Recently, there has been significant theoretical progress towards fixed-parameter algorithms for the DOMINATING SET problem of planar graphs. It is known that the problem on a planar graph with n vertices and dominating number k can be solved in $O(2^{O(\sqrt{k})}n)$ time using tree/branch-decomposition based algorithms. In this paper, we report computational results of Fomin and Thilikos algorithm which uses the branch-decomposition based approach. The computational results show that the algorithm can solve the DOMINATING SET problem of large planar graphs in a practical time and memory space for the class of graphs with small branchwidth. For the class of graphs with large branchwidth, the size of instances that can be solved by the algorithm in practice is limited to about one thousand edges due to a memory space bottleneck. The practical performances of the algorithm coincide with the theoretical analysis of the algorithm. The results of this paper suggest that the branch-decomposition based algorithms can be practical for some applications on planar graphs.

1. Introduction

Given an undirected graph $G(V, E)$, a *k-dominating set* D of G is a subset of k vertices of G such that for every vertex $v \in V(G)$, either $v \in D$ or v is adjacent to a vertex $u \in D$. The *dominating number* of G, denoted by $\gamma(G)$, is the minimum k such that G has a k-dominating set. Given G and an integer k, The DOMINATING SET problem is to decide if $\gamma(G) \leq k$. The optimization version of the problem is to find a dominating set D with $|D| = \gamma(G)$. The DOMINATING SET problem is a core NP-complete problem in combinatorial optimization and graph theory [25]. It also has wide practical applications such as resource allocations [29], domination problems in electric networks [27], and wireless ad hoc networks [42]. The books of Haynes et al. give a survey on the rich literature of algorithms and complexity of the DOMINATING SET problem [28,29]. A recent experimental study on the heuristic algorithms for the DOMINATING SET problem can be found in [39].

The DOMINATING SET problem is NP-hard. Approximation algorithms and exact fixed-parameter algorithms have been extensively studied to tackle the intractability of the problem. A minimization problem P of size n is α-approximable if there is an algorithm which runs in polynomial time in n and produces a solution of P with value at most αOPT, where OPT is the value of the optimal solution of P and $\alpha \geq 1$. If P is $(1 + \epsilon)$-approximable for every fixed $\epsilon > 0$, P is polynomial time approximable (i.e., has a PTAS). Problem P is fixed-parameter tractable if given a parameter k, OPT can be computed in $O(f(k)n^{O(1)})$ time, where $f(k)$ is a computable function in k. For an arbitrary undirected graph G of n

☆ A preliminary version of this paper appeared in the Proc. of the 2nd International Conference on Combinatorial Optimization and Applications (COCOA 2008) [M. Marzban, Q. Gu, X. Jia. Computational study on dominating set problem of planar graphs, in: Proc. of the 2nd International Conference on Combinatorial Optimization and Applications, COCOA 2008, in: LNCS, vol. 5165, 2008, pp. 89–102]

* Corresponding author. Tel.: +1 778 782 6705; fax: +1 778 782 3045.
E-mail addresses: mmarzban@cs.sfu.ca (M. Marzban), qgu@cs.sfu.ca (Q.-P. Gu), csjia@cityu.edu.hk (X. Jia).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.04.012

vertices, the DOMINATING SET problem is known $(1 + \log n)$-approximable [30], but not approximable within a factor of $(1 - \epsilon) \ln n$ for any $\epsilon > 0$ unless $NP \subseteq DTIME(n^{\log \log n})$ [22]. The problem is also known fixed-parameter intractable unless the parameterized complexity classes collapse [20,21]. If the problem is restricted to planar graphs, it is known as the PLANAR DOMINATING SET problem which is still NP-hard [25]. But the PLANAR DOMINATING SET problem is known admitting a PTAS [9] and fixed-parameter tractable [20].

In recent years, there have been significant improvements on the time complexity of algorithms for various fixed-parameter tractable problems in planar graphs. The improvements result in exponential speedups of the algorithms, reducing the running time from $2^{O(k)}n^{O(1)}$ to $2^{O(\sqrt{k})}n^{O(1)}$, for the problems with parameter k. Examples of such subexponential time algorithms include those which solve the PLANAR VERTEX COVER problem, the PLANAR DOMINATING SET problem, and PLANAR LONGEST PATH problem in $O(2^{3.57\sqrt{k}}k+n)$, $O(2^{11.98\sqrt{k}}k+n^3)$, and $O(2^{10.52\sqrt{k}}n+n^3)$ time, respectively [15]. The subexponential algorithms mentioned above use the tree/branch-decompositions of graphs introduced by Robertson and Seymour as a tool [36–38]. More specifically, those algorithms (1) compute a tree or branch decomposition of small width for the input graph and (2) solve the problems by the dynamic programming method based on the found decomposition. There are two key ingredients in realizing the speedups. The first is that a tree or branch decomposition of width $l = O(\sqrt{k})$ can be computed in $O(n^{O(1)})$ time for those problems with parameter k and the second is that Step (2) can be computed in $2^{O(l)}n^{O(1)}$ time. It is not difficult to realize Step (2) in $2^{O(l)}n^{O(1)}$ time for a class of problems represented by the vertex cover and dominating set problems, where global infomation is not needed in the dynamic programming. For a class of problems represented by the longest path problem and connected dominating set problem, where some global information is required in the dynamic programming, Dorn et al. give an approach based on the sphere cut branch decompositions and non-crossing partitions to realize Step (2) in $2^{O(l)}n^{O(1)}$ time [19].

Demaine et al. introduce a new framework for designing fixed-parameter algorithms with $2^{O(\sqrt{k})}n^{O(1)}$ running time [13]. This framework builds on the bidimensionality theory and can be applied to extend the subexponential time fixed-parameter algorithms for the class of problems including vertex cover and dominating set problems in planar graphs to a broader class of graphs excluding a fixed graph as a minor. The work of [17] further makes such extensions possible for the non-local problems represented by the longest path and connected dominating set problems. The survey papers of [14,18] give a summary on the progresses of the subexponential fixed-parameter algorithms, the bidimensionality theory and its algorithmic applications.

The PLANAR DOMINATING SET problem has been extensively studied. First fixed-parameter algorithms for the problem on graphs with $\gamma(G) = k$ have running time $O(11^k n)$ [20] and $O(8^k n)$ [7]. The exponential speedups give algorithms for the problem with running time $O(2^{c\sqrt{k}}n)$, where c is some constant [4,23,31]. Most of the subexponential algorithms use a tree-decomposition based algorithm: For a planar graph G with $\gamma(G) = k$, a tree decomposition of width $b\sqrt{k}$, b is a constant, is computed and the dynamic programming part runs in $O(2^{2b\sqrt{k}}n)$ time [4]. One problem with those algorithms is that the constant $c = 2b$ is too large for solving the PLANAR DOMINATING SET problem in practice. Instead of a tree decomposition, a branch decomposition can be used in the above dynamic programming algorithms for the PLANAR DOMINATING SET problem. Fomin and Thilikos give such an algorithm (called FT Algorithm in what follows) of running time $O(2^{(3\log_4 3)bw(G)}k + n^3)$, where $bw(G)$ is the branchwidth of G [23]. Fomin and Thilikos prove that $bw(G) \leq 3\sqrt{4.5k}$ and $O(2^{(3\log_4 3)bw(G)}) = O(2^{15.13\sqrt{k}})$, reducing the constant c to 15.13 [23,24]. Dorn proposes an approach of applying the distance product of matrices to the dynamic programming step in branch/tree-decomposition based algorithms for the problem [15]. If a conventional $O(n^3)$ time algorithm is used for the distance product of matrices, this approach has the same constant $c = 15.13$ as that of FT Algorithm. It is known that the distance product of integer matrices can be realized by the fast matrix multiplication [44]. If the distance product of matrices is realized by the $O(n^\omega)$ $(\omega < 2.376)$ time fast matrix multiplication method [12], the constant c is improved to 11.98. However, the constant hidden in the Big-Oh may be huge when the fast matrix multiplication is used. Dorn also proves that the PLANAR DOMINATING SET problem can be solved in $O(3^{tw(G)}n^{O(1)})$ time, where $tw(G)$ is the treewidth of G [16]. The tree decomposition used in Dorn's proof is closely related to the branch decomposition and the algorithm of [16] has the same sublinear exponent in the time complexity as that of the algorithm in [15]. An encouraging fact on branch decomposition is that an optimal branch decomposition of a planar graph can be computed in polynomial time [26,41]. This makes the branch-decomposition based algorithms receiving increasing attention for the problems on planar graphs.

Another important progress on the algorithmic tractability of the PLANAR DOMINATING SET problem is that the problem is shown having a linear size kernel [8]. More specifically, Alber et al. give an $O(n^3)$ time algorithm which, given a planar graph G with $\gamma(G) = k$, produces a reduced graph H (kernel) such that H has $O(k)$ vertices, $\gamma(H) = k' \leq k$, and a minimum dominating set of G can be constructed from a minimum dominating set of H in linear time [8]. In general, H and k' are smaller than G and k, respectively, since in the reduction process, a number of vertices in a minimum dominating set of H have been decided. This reduction process reduces the sublinear exponent from $c\sqrt{k}$ to $c\sqrt{k'}$ and thus improves the running time of the fixed-parameter algorithms for the PLANAR DOMINATING SET problem. This result is used in FT Algorithm which has three major steps [23]: Step I computes a kernel H of G by the data reduction process of [8] in $O(n^3)$ time. Step II finds an optimal branch decomposition of H with width $bw(H)$. This can be done by algorithms of [10,26] in $O(k^3)$ time. Step III computes a minimum dominating set D' of H using the dynamic programming method based on the branch decomposition in $O(2^{(3\log_4 3)bw(H)}k)$ time and constructs a minimum dominating set D of G from D' in linear time. It is proved in [23,24] that

the branchwidth $bw(H) \leq 3\sqrt{4.5k'}$ and FT Algorithm has time complexity $O(2^{15.13\sqrt{k'}}k+n^3)$. The memory space required in Step III is $O(3^{bw(H)}k) = O(2^{10.1\sqrt{k'}}k)$ bytes for computing the dominating number $\gamma(H)$ only and $O(2^{10.1\sqrt{k'}}k^2)$ for computing a minimum dominating set D' of H. Alber et al. report that the data reduction computes a much smaller kernel in practice for a class of planar graphs [3,8]. Recently, Bian et al. report that an optimal branch decomposition of a planar graph can be computed efficiently in practice [10,11]. These results provide the base for testing the practical efficiency of FT Algorithm for the PLANAR DOMINATING SET problem.

Although significant theoretical progresses have been made towards the fixed-parameter algorithms for the PLANAR DOMINATING SET problem, limited work has been done on the practical performances of these algorithms. Alber et al. give experimental evaluations on the tree-decomposition based algorithms for several problems including the vertex cover and dominating set problems in planar graphs [5,6]. The experimental studies are performed on a class of random maximal planar graphs and their subgraphs generated by LEDA [2,34] and the results for the PLANAR DOMINATING SET problem can be found in [5]. In this paper, we report the computational study on FT Algorithm for the PLANAR DOMINATING SET problem. In our implementations of FT Algorithm, in addition to the data reduction rules of [3,8], we introduce new data reduction rules and use the recent works on planar branch decompositions. The new data reduction rules further reduce the kernel size and improve the running time of FT Algorithm. We have tested our implementations of FT Algorithm on several classes of planar graphs, including the random maximal planar graphs and their subgraphs from LEDA [2,34], Delaunay triangulations of point sets taken from TSPLIB [35], triangulations and intersection graphs of segments from LEDA, Gabriel graphs, and planar graphs from PIGALE library [1]. The computational results show that the size of instances that can be solved in a practical time and memory space mainly depends on the branchwidth of the kernels. For example, as shown in the Appendix, the random maximal planar graphs and their subgraphs have branchwidth at most four. This class of graphs are used as the test instances in previous studies for the data reduction [3,8] and the tree-decomposition based algorithms for the PLANAR DOMINATING SET problem [5]. Step I reduces the problem size significantly and the PLANAR DOMINATING SET problem can be solved in about twenty minutes for instances of size about forty thousand edges in this class. On the other hand, for Delaunay triangulation and Gabriel graphs, because the branchwidth of kernels increases fast in instance size, the size of instances that can be solved in a practical time and memory space is limited to about one thousand edges (the memory space required in Step III is a bottleneck for solving instance of large branchwidth). For triangulation graphs, intersection graphs, and graphs from PIGALE library, the branchwidth of kernels increases slowly or does not grow in instance size. Large instances of size about thirty thousand edges in these classes can be solved in a practical time and memory space. These results coincide with the theoretical analysis of FT Algorithm [23]: it runs and requires memory space exponentially in the branchwidth of the kernel of size $O(k)$ and $k \geq b(bw(G))^2$ for some constant b. Because the kernel of G has $O(k)$ vertices, the analysis suggests that a large branchwidth of the instance implies a large kernel, Step I may not reduce the problem size much, and the kernel may have a large branchwidth. For a kernel H with large branchwidth, FT Algorithm is not practical because Step III of the algorithm runs and requires memory space exponentially in the branchwidth of H. Compared with the previous study of [5], our study is conducted over a much broader range of graphs with much larger sizes. For the same class of graphs used in both studies, the results suggest that FT Algorithm is more efficient than the algorithm used in [5].

We also report the computational results of using distance product of matrices proposed by Dorn in Step III of FT Algorithm. When a conventional $O(n^3)$ time distance product of matrices is used, Dorn's algorithm has the same sublinear exponent in the time complexity as that of FT Algorithm and the practical performances of the two algorithms are very similar as well. Theoretically, the sublinear exponent of FT Algorithm can be improved when the fast matrix multiplication is used for realizing the distance product of matrices in Step III. However, for the distance product of matrices in practice, the fast matrix multiplication method is slower and requires more memory space than the conventional $O(n^3)$ time method.

The results of this paper give a concrete example on using branch-decomposition based algorithms for solving important hard problems in planar graphs and show that the PLANAR DOMINATING SET problem can be solved in practice for some applications. This work may bring the theory of branch decomposition closer to practice.

The rest of the paper is organized as follows: In the next section, we review FT Algorithm and the approach of applying the distance product of matrices in the dynamic programming step. We introduce the data reduction rules in Section 3. Computational results are reported in Section 4. The final section concludes the paper.

2. Fomin and Thilikos algorithm

We first introduce some definitions and terminology. Readers may refer to a textbook on graph theory (e.g., the one by West [43]) for basic definitions and terminology on graphs. In this paper, graphs are undirected unless otherwise stated. Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. A *branch decomposition* of G is a tree T_B such that the set of leaves of T_B is $E(G)$ and each internal node of T_B has node degree three. For each link e of T_B, removing e separates T_B into two subtrees. Let E' and E'' be the sets of leaves of the subtrees. Let S_e be the set of vertices of G incident to both an edge of E' and an edge of E''. The width of e is $|S_e|$ and the width of T_B is the maximum width of all links of T_B. The *branchwidth* $bw(G)$ of G is the minimum width of all branch-decompositions. We call a link $e = \{x, y\}$ a leaf link if one of x and y is a leaf node of T_B, otherwise an internal link. Notice that S_e is a set which separates G into two subgraphs induced by edges of E' and E'', respectively.

We say a vertex u is dominated by a vertex v if u and v are adjacent. A vertex set U is dominated by a vertex set V if for every vertex $u \in U$ there is a vertex $v \in V$ such that u and v are adjacent or $u \in V$. Given two graphs G and H, we say

$size(H) \leq size(G)$ if $|V(H)| \leq |V(G)|$ and $|E(H)| \leq |E(G)|$. In the rest of the paper, the PLANAR DOMINATING SET problem is used for the optimization version of the problem unless otherwise stated.

Now we briefly review FT Algorithm. Readers may refer to [23] for more details. FT Algorithm solves the PLANAR DOMINATING SET problem of G in three steps. Step I computes a kernel H of G by the data reduction process such that $size(H) \leq size(G)$, $\gamma(H) \leq \gamma(G)$, and a minimum dominating set D of G can be computed from a minimum dominating set D' of H in linear time. Step II finds an optimal branch decomposition T_B of H. Step III computes a minimum dominating set D' of H using the dynamic programming method based on T_B and constructs a minimum dominating set D of G from D'.

In Step I, the data reduction rules introduced in [8] are used to decide if some vertices of G can be included into D or excluded for computing D. If a vertex v has been decided to be included in D, v is colored black. If v has been decided to be excluded for computing D in the future, v is removed from G. Every other vertex is colored grey. After the reduction process, we get a kernel $H(B \cup C, E)$, where B and C are the sets of black and grey vertices, respectively. The specific reduction rules will be introduced in the next section.

To compute an optimal branch decomposition T_B of H, either the edge-contraction algorithms [26,41] or the divide-and-conquer algorithms [10] can be used. The divide-and-conquer algorithms are faster for large graphs in practice.

In Step III, given a kernel $H = (B \cup C, E)$, we find a minimum $D' \subseteq (B \cup C)$ such that $D' \supseteq B$ and D' dominates all vertices of C. As shown later, a minimum dominating set D of G can be constructed from D' in linear time. To compute D', first the branch decomposition T_B of H is converted into a rooted binary tree by replacing a link $\{x, y\}$ of T_B by three links $\{x, z\}$, $\{z, y\}$, and $\{z, r\}$, where z and r are new nodes to T_B, r is the root, and $\{z, r\}$ is an internal link. For every internal link e of T_B, e has two children links incident to e. For every link e of T_B, let T_e be the subtree of T_B consisting of all descendant links of e. Let H_e be the subgraph of H induced by the edges at leave nodes of T_e. To compute a minimum dominating set D' of H, we find all dominating sets (solutions) of H_e from which D' may be constructed for every link e of T_B by a dynamic programming method: the solutions of H_e for each leaf link e is computed by enumeration and the solutions for an internal link e is computed by merging the solutions for the children links of e. To find a solution of H_e, each vertex of S_e is colored by one of the following colors.

Black denoted by 1, meaning that the vertex is included in the dominating set.
White denoted by 0, meaning that the vertex is dominated at the current step of the algorithm and is not in the dominating set.
Grey denoted by $\hat{0}$, meaning that we have not decided to color the vertex into black or white yet at the current step.

A solution of H_e subject to a coloring $\lambda \in \{0, \hat{0}, 1\}^{|S_e|}$ is a minimum set $D_e(\lambda)$ satisfying

- for $u \in B \cap S_e$, $\lambda(u)$ is black;
- every vertex of $V(H_e) \setminus S_e$ is dominated by a vertex of $D_e(\lambda)$; and
- for every vertex $u \in S_e$ if $\lambda(u)$ is black then $u \in D_e(\lambda)$, if $\lambda(u)$ is white then $u \notin D_e(\lambda)$ and u is dominated by a vertex of $D_e(\lambda)$.

Intuitively, $D_e(\lambda)$ is a minimum set to dominate the vertices of H_e with grey vertices removed, subject to the condition that the vertices of S_e are colored by λ. For every coloring $\lambda \in \{0, \hat{0}, 1\}^{|S_e|}$, $a_e(\lambda)$ is defined as $|D_e(\lambda)|$ if there is a solution of H_e subject to λ, otherwise as $+\infty$.

For a leaf link e, colorings λ and sets $D_e(\lambda)$ are computed by enumeration. Assume that an internal link e has children links e_1 and e_2 in T_B. The colorings λ of S_e and sets $D_e(\lambda)$ are computed from the colorings λ_1 of S_{e_1}, sets $D_{e_1}(\lambda_1)$, colorings λ_2 of S_{e_2}, and sets $D_{e_2}(\lambda_2)$. Let $X_1 = S_e \setminus S_{e_2}$, $X_2 = S_e \setminus S_{e_1}$, $X_3 = S_e \cap S_{e_1} \cap S_{e_3}$, and $X_4 = (S_{e_1} \cup S_{e_2}) \setminus S_e$. A coloring λ of S_e is formed from λ_1 and λ_2 if:

(1) For $u \in X_1$, $\lambda(u) = \lambda_1(u)$.
(2) For $u \in X_2$, $\lambda(u) = \lambda_2(u)$.
(3) For $u \in X_3$, if $\lambda_1(u) = \lambda_2(u) = 1$ then $\lambda(u) = 1$; if $\lambda_1(u) = \lambda_2(u) = \hat{0}$ then $\lambda(u) = \hat{0}$; and if $\lambda_1(u) = 0$ and $\lambda_2(u) = \hat{0}$, or $\lambda_1(u) = \hat{0}$ and $\lambda_2(u) = 0$ then $\lambda(u) = 0$.
(4) For $u \in X_4$, $\lambda_1(u) = \lambda_2(u) = 1$, or $\lambda_1(u) = 0$ and $\lambda_2(u) = \hat{0}$, or $\lambda_1(u) = \hat{0}$ and $\lambda_2(u) = 0$.

For a coloring λ of S_e formed from λ_1 and λ_2, the minimum dominating set $D_e(\lambda)$ is the minimum set among the sets of $D_{e_1}(\lambda_1) \cup D_{e_2}(\lambda_2)$. For $e = \{z, r\}$, a minimum set $D_e(\lambda)$ among all colorings λ of S_e is a minimum dominating set of H.

Notice that the original description of FT Algorithm for Step III in [23] does not have the part for handling the vertices colored black in data reduction process. We have added this part and our description is slightly different from the original one.

In the implementation of FT Algorithm, we put $a_{e_1}(\lambda_1)$ (and a pointer to $D_{e_1}(\lambda_1)$) in a table T_1. Similarly, we put $a_{e_2}(\lambda_2)$ in a table T_2. Table T_1 has at most $3^{|S_{e_1}|} = 3^{|X_1|+|X_3|+|X_4|}$ entries and Table T_2 has at most $3^{|S_{e_2}|} = 3^{|X_2|+|X_3|+|X_4|}$ entries. We use the following index for the entries of T_1 and T_2: The entries of T_1 are first partitioned into $3^{|X_1|}$ groups by the colors of the vertices in X_1. Similarly the entries of T_2 are partitioned into $3^{|X_2|}$ groups. The entries of each table within each group is further identified by the colors of the vertices in $X_3 \cup X_4$. To find a minimum $D_e(\lambda)$ from $D_{e_1}(\lambda_1)$ and $D_{e_2}(\lambda_2)$, we first compute

$$a_e(\lambda) = \min_{\lambda_1, \lambda_2 form \lambda} \{a_{e_1}(\lambda_1) + a_{e_2}(\lambda_2) - \#_1(X_3 \cup X_4, \lambda_1)\},$$

where $\#_1(X_3 \cup X_4, \lambda_1)$ is the number of vertices in $X_3 \cup X_4$ taking color 1 in λ_1. The colors of λ_1's and λ_2's which form λ are the entries in the group of T_1 and the entries in the group of T_2 identified by the colors of λ for vertices of X_1 and X_2, respectively. Then, a corresponding minimum $D_e(\lambda)$ is computed. The results of $a_e(\lambda)$ are kept in a table T of at most $3^{|S_e|} = 3^{|X_1|+|X_2|+|X_3|}$ entries. The memory space required for computing table T and $D_e(\lambda)$ is $O((3^{|S_e|} + 3^{|S_{e_1}|} + 3^{|S_{e_2}|})k) = O(3^{bw(H)}k)$ because $\max\{|S_e|, |S_{e_1}|, |S_{e_2}|\} \leq bw(H)$ and $\max\{|D_e(\lambda)|, |D_{e_1}(\lambda_1)|, |D_{e_2}(\lambda_2)|\} \leq |V(H)| = O(k)$. If we only compute $\gamma(H)$ then we only need to compute table T and the required memory space is $O(3^{bw(H)})$. Since there are $O(k)$ links in the branch decomposition T_B, the total memory space required in Step III is $O(3^{bw(H)}k^2)$ for computing a minimum dominating set and $O(3^{bw(H)}k)$ for computing the dominating number of H. We call the above *index method*.

Dorn proposes using distance product of matrices to compute the minimum $D_e(\lambda)$ for all colorings λ [15]. We also implemented this approach (*distance product method*): The entries of Table T_1 are arranged into $r = 3^{|X_3|}$ matrices $A_1, \ldots, A_r$ of $3^{|X_1|}$ rows and $3^{|X_4|}$ columns ($|X_3| \leq 2$ for a planar graph with a fixed embedding and a branch decomposition found by the algorithms used in this paper [15]). The entries of Table T_2 are arranged into r matrices $B_1, \ldots, B_r$ of $3^{|X_4|}$ rows and $3^{|X_2|}$ columns. Each row of A_l ($1 \leq l \leq r$) is identified by a sequence of $|X_1|$ colors and each column of B_l is identified by a sequence of $|X_2|$ colors, with each color from $\{0, \hat{0}, 1\}$. Each column of A_l (each row of B_l) is identified by a sequence of $|X_4|$ colors. We arrange the columns of A_l in the increasing alphabetic order, defined by $0 < \hat{0} < 1$, of the color sequences. We arrange the rows of B_l in the increasing alphabetic order, defined by $\hat{0} < 0 < 1$, of the color sequences. We define a one-to-one mapping between the colorings of $\{0, \hat{0}, 1\}^{|X_3|}$ and $1, 2, \ldots, r$ based on the alphabetic order defined by $0 < \hat{0} < 1$. The value $a_{e_1}(\lambda_1)$ in each element of A_l ($1 \leq l \leq r$) is changed to $a_{e_1}(\lambda_1) - \#_1(X_3 \cup X_4, \lambda_1)$. For every l with $1 \leq l \leq r$, the distance product $C_l = A_l \times B_l$ is computed, where C_l is a matrix of $|X_1|$ rows and $|X_2|$ columns, and $C_l[i, j] = \min\{A_l[i, k]+B_l[k, j], 1 \leq k \leq 3^{|X_4|}\}$. We say a coloring l of $\{0, \hat{0}, 1\}^{|X_3|}$ is formed by colorings l_1 and l_2 of $\{0, \hat{0}, 1\}^{|X_3|}$ if

- For $u \in X_3$, if $l_1(u) = l_2(u) = 1$ then $l(u) = 1$; if $l_1(u) = l_2(u) = \hat{0}$ then $l(u) = \hat{0}$; and if $l_1(u) = 0$ and $l_2(u) = \hat{0}$, or $l_1(u) = \hat{0}$ and $l_2(u) = 0$ then $l(u) = 0$.

Every element $C_l[i, j]$ is further updated by $\min\{C_{l_1}[i, j], \ldots, C_{l_p}[i, j]\}$, where $l_1, \ldots, l_p$ are the colorings of $\{0, \hat{0}, 1\}^{|X_3|}$ which form coloring l.

If a conventional $O(n^3)$ time algorithm is used for the distance product of the matrices, Step III takes $(2^{(3\log_4 3)bw(H)}k) = O(2^{15.13\sqrt{k'}}k)$ time, and requires $O(3^{bw(H)}k^2)$ and $O(3^{bw(H)}k)$ memory spaces for computing a minimum dominating set and $\gamma(H)$ of H, respectively, the same as those of the index method. If the distance product of matrices is realized by the $O(n^\omega)$ ($\omega < 2.376$) time fast matrix multiplication method, Step III has time complexity $O(2^{11.98\sqrt{k'}}n^{O(1)})$. In practice, the fast matrix multiplication method is slower due to the big hidden constant behind the Big-Oh than the conventional distance product method. The fast matrix multiplication method also requires more memory space due to the recursive computation.

3. Data reduction

In this section, we introduce the data reduction rules used in our implementation of FT Algorithm for Step I. All reduction rules of [3,8] are used. To enhance the data reduction effect, we also propose some new reduction rules. Following the convention of FT Algorithm, we color each vertex of G by black or grey, and may remove some vertices from G by those reduction rules. After the data reduction step, we get a kernel $H(B \cup C, E)$, recall that B and C are the sets of black and grey vertices, respectively. For a vertex v, let $N(v) = \{u|\{u, v\} \in E(G)\}$, $N[v] = N(v) \cup \{v\}$, $B(v) = B \cap N(v)$, and $C(v) = C \cap N(u)$. For a set U of vertices, let $N(U) = \cup_{v \in U} N(v)$. For a vertex u, if there is a black vertex $v \in N[u]$, we mark u *dominated*. Initially, every vertex of G is unmarked. In the data reduction step, some vertices are marked. Let X be the set of marked vertices and Y be the set of unmarked vertices. For $v \in V(G)$, the following is introduced in [8]:

$$N_1(v) = B(v) \cup \{u|u \in C(v), N(u) \setminus N[v] \neq \emptyset\},$$
$$N_2(v) = \{u|u \in N(v) \setminus N_1(v), N(u) \cap N_1(v) \neq \emptyset\}, \text{ and}$$
$$N_3(v) = N(v) \setminus (N_1(v) \cup N_2(v)).$$

Rule 1 ([8]). For $v \in V(G)$, if $N_3(v) \cap Y \neq \emptyset$ then remove $N_2(v)$ and $N_3(v)$ from G, color v black, and mark $N[v]$ dominated.

For a pair of vertices $v, w \in V(G)$, let $N(v, w) = N(v) \cup N(w) \setminus \{v, w\}$, $B(v, w) = B \cap N(v, w)$, $C(v, w) = C \cap N(v, w)$, and $N[v, w] = N[v] \cup N[w]$. The following is introduced in [8]:

$$N_1(v, w) = B(v, w) \cup \{u|u \in C(v, w), N(u) \setminus N[v, w] \neq \emptyset\},$$
$$N_2(v, w) = \{u|u \in N(v, w) \setminus N_1(v, w), N(u) \cap N_1(v, w) \neq \emptyset\},$$
$$N_3(v, w) = N(v, w) \setminus (N_1(v, w) \cup N_2(v, w)).$$

Rule 2 ([8]). For $v, w \in V(G)$ with both v and w grey, assume that $|N_3(v, w) \cap Y| \geq 2$ and $N_3(v, w) \cap Y$ can not be dominated by a single vertex of $N_2(v, w) \cup N_3(v, w)$.

Case 1: $N_3(v, w) \cap Y$ can be dominated by a single vertex of $\{v, w\}$.

- (1.1) If $N_3(v, w) \cap Y \subseteq N(v)$ and $N_3(v, w) \cap Y \subseteq N(w)$ then remove $N_3(v, w)$ and $N_2(v, w) \cap N(v) \cap N(w)$ from G and add new gadget vertices z and z' with edges $\{v, z\}$, $\{w, z\}$, $\{v, z'\}$, and $\{w, z'\}$ to G.
- (1.2) If $N_3(v, w) \cap Y \subseteq N(v)$ but $N_3(v, w) \cap Y \not\subseteq N(w)$ then remove $N_3(v, w)$ and $N_2(v, w) \cap N(v)$ from G, color v black, and mark $N[v]$ dominated.
- (1.3) If $N_3(v, w) \cap Y \subseteq N(w)$ but $N_3(v, w) \cap Y \not\subseteq N(v)$ then remove $N_3(v, w)$ and $N_2(v, w) \cap N(w)$ from G, color w black, and mark $N[w]$ dominated.

Case 2: If $N_3(v, w) \cap Y$ can not be dominated by a single vertex of $\{v, w\}$ then remove $N_2(v, w)$ and $N_3(v, w)$ from G, mark v and w black, and mark $N[v, w]$ dominated.

In Rule 1 and Rule 2 (Cases 1.2, 1.3, and 2) of [8], gadget vertices are used to guarantee some vertices to be included in the solution set. In [3] the rules are implemented in a way that the vertices to be included in the solution set are removed. Our descriptions are slightly different from the previous ones: we do not use gadget vertices nor remove the vertices to be included to the solution set but color them black. Our descriptions allow us to have new reduction rules given below that may further reduce the size of the kernel.

Rule 3.

3.1: For $v, w \in V(G)$ with v black and w grey, if $(N_3(v, w) \cap Y) \setminus N(v) \neq \emptyset$ then remove $N_2(v, w) \cup N_3(v, w)$, color w black, and mark $N[w]$ dominated; otherwise remove $(N_2(v, w) \cup N_3(v, w)) \cap N(v)$.

3.2: For $v, w \in V(G)$ with v grey and w black, if $(N_3(v, w) \cap Y) \setminus N(w) \neq \emptyset$ then remove $N_2(v, w) \cup N_3(v, w)$, color v black, and mark $N[w]$ dominated; otherwise remove $(N_2(v, w) \cup N_3(v, w)) \cap N(w)$.

3.3: For $v, w \in V(G)$ with both v and w black, remove $N_2(v, w) \cup N_3(v, w)$.

Lemma 3.1. *Given a graph G, let G' be the graph obtained by applying Rule 3 for $v, w \in V(G)$. Then size$(G') \leq$ size(G), $\gamma(G') \leq \gamma(G)$, and a minimum dominating set D' of G' that contains all black vertices of G' is a minimum dominating set of G that contains all black vertices of G.*

Proof. For $v, w \in V(G)$ with v black and w grey, assume that $(N_3(v, w) \cap Y) \setminus N(v) \neq \emptyset$. For $u \in (N_3(v, w) \cap Y) \setminus N(v)$ and x which dominates $u, x \in \{w\} \cup N_2(v, w) \cup N_3(v, w)$. Since $N(N_2(v, w) \cup N_3(v, w)) \subseteq N[v] \cup N[w]$, we should include w into D to dominate $(N_3(v, w) \cap Y) \setminus N(v)$. Therefore, we can remove $N_2(v, w) \cup N_3(v, w)$ from G. Assume that $(N_3(v, w) \cap Y) \setminus N(v) = \emptyset$. For $u \in (N_2(v, w) \cup N_3(v, w)) \cap N(v)$, u is dominated by v and $N(v) \cup N(u) \subseteq N(v) \cup N(w)$. This implies that we can at least include w rather than u to get D. At this point, we can not decide if we should include w into D or not because there might be a vertex x with $N(w) \subseteq N(x)$ that should be included in D. But we can exclude $(N_2(v, w) \cup N_3(v, w)) \cap N(v)$ from D. Since $(N_2(v, w) \cup N_3(v, w)) \cap N(v)$ is dominated by v, we can remove $(N_2(v, w) \cup N_3(v, w)) \cap N(v)$ from G. This completes the proof for (3.1).

The proof for (3.2) is a symmetric argument of that for (3.1).

For $v, w \in V(G)$ with both v and w black, since $N(N_2(v, w) \cup N_3(v, w)) \subseteq N[v] \cup N[w]$, we can remove $N_2(v, w) \cup N_3(v, w)$ from G. □

Rule 4 ([3]).

4.1: Delete edges between vertices of X (vertices marked dominated).

4.2: If $u \in X$ has $|C(u)| \leq 1$ then remove u.

4.3: For $u \in X$ with $C(u) \cap Y = \{u_1, u_2\}$, if u_1 and u_2 are connected by a path of length at most 2 then remove u.

4.4: For $u \in X$ with $C(u) \cap Y = \{u_1, u_2, u_3\}$, if $\{u_1, u_2\}, \{u_2, u_3\} \in E(G)$ then remove u.

To perform the data reduction, we first apply Rule 1 for every vertex of G. Next for every pair of vertices v and w of G, we apply either Rule 2 or Rule 3 depending on the colors of v and w. Then we apply Rule 4. We repeat the above until Rules 1–4 do not change the graph. From the results of [8,3] on Rules 1, 2, and 4, and Lemma 3.1, we have the following result.

Theorem 3.1. *Given a planar graph G, let $H(B \cup C, E)$ be the kernel obtained by applying the reduction rules described above and D' be a minimum vertex set of $H(B \cup C, E)$ such that $D' \supseteq B$ and D' dominates C. Then a minimum dominating set D of G can be constructed from D' in linear time.*

In our implementation of FT Algorithm, $D' = D$. Since the vertices of B are included to D' in Step I, the number of vertices to be included in D' in Step III is $|D| - |B|$. Therefore, the size of the dominating set for the kernel decided in Step III is actually $k' = \gamma(G) - |B|$. If Step I gives an non-empty set B of black vertices, k' is smaller than $k = \gamma(G)$.

Given a planar graph G, let $H(B \cup C, E)$ be the kernel obtained from Step I, T_B be an optimal branch decomposition of H, and $l(H) = \max\{|C \cap S_e|, e \in E(T_B)\}$. It is shown in [8] that $H(B \cup C, E)$ can be computed in $O(n^3)$ time. T_B can be computed by either the edge-contraction algorithm [26] or a divide-and-conquer algorithm [10] in $O(|E(H)|^3)$ time. It is shown in [23] that Step III has time complexity $O(2^{(3 \log_4 3) l(H)} |E(H)|)$. Therefore, FT Algorithm takes $O(2^{(3 \log_4 3) l(H)} |E(H)| + n^3)$ time to solve the PLANAR DOMINATING SET problem. Notice that $l(H) \leq bw(H)$ and in what follows, we use $l(H)$ for the branchwidth of kernel H.

4. Computational results

We implemented FT Algorithm and tested our implementations on six classes of planar graphs from some libraries including LEDA [2,34] and PIGALE [1]. LEDA generates two types of planar graphs. One type of graphs are the random maximal planar graphs and their subgraphs and the other type of graphs are the planar graphs based on some geometric properties, including the Delaunay triangulations and triangulations of points, and the intersection graphs of segments, uniformly distributed in a two-dimensional plane. Instances of Class (1) are the random maximal graphs and their subgraphs generated by LEDA. This class of instances have been used by Alber et al. in their studies on the data reduction rules used in Step I [3,8] and the tree-decomposition based subexponential algorithms for the vertex cover and dominating set problems [5,6]. Instances of Class (2) are Delaunay triangulations of point sets taken from TSPLIB [35]. Instances of Classes (3) and (4) are the triangulations and intersection graphs generated by LEDA, respectively. Instances of Class (5) are Gabriel graphs of the points uniformly distributed in a two-dimensional plane. Instances of Classes (2)–(5) are graphs based on some geometric properties. The DOMINATING SET problem on those graphs has important applications such as the virtual backbone design of wireless networks [32]. Instances of Class (6) are random planar graphs generated by the PIGALE library [1]. PIGALE provides a number of planar graph generators. We used a function in the PIGALE library that randomly generates one of all possible 2-connected planar graphs with a given number of edges based on the algorithms of [40].

Step I of FT Algorithm is implemented as described in the previous section. To compute an optimal branch decomposition T_B, we use the divide-and-conquer algorithm [10]. For Step III, both the index and distance product methods are used. To save memory, we compute the colorings λ and sets $D_e(\lambda)$ for each link e of T_B in the postorder. Once the colorings λ and sets $D_e(\lambda)$ are computed for a link e, the solutions for the children links of e are discarded. The computer used for testing has an AMD Athlon(tm) 64 X2 Dual Core Processor 4600+ (2.4 GHz) and 3 Gbyte memory. The operating system is SUSE Linux 10.2 and the programming language used is C++.

We report the computational results for finding both $\gamma(G)$ and a minimum dominating set of G by FT Algorithm with the index method in Step III in Table 1.[1] For Step I, we give the number $|B|$ of vertices of an optimal dominating set decided in the data reduction and the running time of the step. For Step II, we give the size $|E(H)|$ and branchwidth $l(H) = \max\{|C \cap S_e|, e \in E(T_B)\}$ of kernel H, and the running time of the step. For Step III, we give the dominating number $\gamma(G)$ obtained by FT Algorithm and the running time of the step. The running time is in seconds, and Steps I, II, and III have time complexities $O(|E(G)|^3)$, $O(|E(H)|^3)$, and $O(2^{(3\log_4 3)l(H)}|E(H)|)$, respectively. We use the number of edges to express the size of an instance or a kernel. For each instance size, we have tested three graphs of similar size for Classes (1) and (3)–(6), and Table 1 contains the graph with the worst case running time. Notice that multiple graphs of similar size in Class (2) are not available. The average performance of FT Algorithm over three graphs for some large instances is given in Table 3. We also report the running time of FT Algorithm for computing $\gamma(G)$ only for some large instances in Table 1. For those instances (marked with an "*") FT Algorithm can not compute a minimum dominating set by the computer used in this study because it requires more than 3 GByte memory space but can compute the $\gamma(G)$.

As shown in the Appendix, the instances of Class (1) have branchwidth at most four. These instances have small kernels and Step I is very effective. For the instances included in the table, $|B|$ is very close to $\gamma(G)$ (i.e., Step I finds most vertices in an optimal dominating set) and the kernels are much smaller than the original instances. For some smaller instances not reported in the table, Step I already finds optimal dominating sets. Because the kernels have small size and branchwidth, FT Algorithm is efficient for the instances in this class, for example, an optimal dominating set can be computed for large instances of size up to about 40,000 edges in about 20 min. It is reported in [5] that instances of size about 6000 edges can be solved in about 30 min by a tree-decomposition based algorithm on a computer with a CPU of 750 MHz and 720 MBytes memory space. These results suggest that FT Algorithm is more efficient than the algorithm used in [5] for the graphs in this class.

For Classes (2) and (5), the branchwidth of instances increases fast in instance size (e.g., Class (2) instances rd400 of 1183 edges and u2152 of 6312 edges have branchwidth 17 and 31, respectively, Class (5) instances Gab800 of 1533 edges and Gab2000 of 3911 edges have branchwidth 16 and 26, respectively). For the instances tested, the kernel H of an instance G has the same branchwidth and same size as or only slightly smaller than those of G. The computation time increases significantly when the branchwidth of the kernels increases. This coincides with the theoretical time complexity of FT Algorithm which runs exponentially in $l(H)$.

For Classes (3) and (4), the branchwidth of instances increases slowly in instance size. The data reduction is effective for instances in these classes. For most instances, the kernel size is at most half of the instance size and the branchwidth of the kernel is usually smaller than that of the instance as well. Our data show that the minimum dominating set can be found for instances of size up to about thirty thousand edges in a practical time and memory space. For large instances, the size $|E(H)|$ of kernel H is also important to the running time of Step III. For example, FT Algorithm takes more time to solve Instance rand15000 than that for rand10000. The time difference comes from the differences of both $l(H)$ and $|E(H)|$. For Class (6), the branchwidth of instances does not grow in the instance size. FT Algorithm is efficient for the instances in this class.

[1] Compared with the data in the preliminary version of this paper [33], the running time of FT Algorithm is improved due to an improvement in the implementation of Step III.

Table 1
Computational results (time in seconds) of FT Algorithm with the index method in Step III for instances of Classes (1)–(6). For the instances marked with "*", the time is for computing the dominating number only because the 3 GByte memory is not enough for computing a minimum dominating set.

Class	Graph	$\|E(G)\|$	$bw(G)$	Step I		Step II			Step III		Total
	G			$\|B\|$	Time	$\|E(H)\|$	$l(H)$	time	$\gamma(G)$	Time	Time
(1)	max1500	3 860	4	228	12	78	3	<1	236	<1	12
	max6000	7 480	4	2214	55	32	2	<1	2219	<1	55
	max8000	13 395	4	2186	336	194	3	<1	2211	<1	337
	max11000	28 537	4	1679	799	208	4	1	1695	<1	800
	max13500	38 067	4	1758	1203	302	3	1	1779	<1	1 204
(2)	kroB150	436	10	0	<1	436	10	<1	23	10	10
	pr226	586	7	12	1	126	6	<1	21	<1	1
	pr299	864	11	1	<1	824	11	1	47	35	37
	tsp225	622	12	0	<1	622	12	1	37	109	110
	a280	788	13	1	<1	730	13	1	43	336	337
(3)	tri2000	5 977	8	136	57	3 192	7	140	321	1	198
	tri4000	11 969	9	252	256	6 888	7	1641	653	6	1 903
	tri6000	17 979	9	312	566	11 691	8	2991	975	19	3 576
	tri8000	23 975	9	497	830	13 524	7	6900	1283	20	7 750
	tri10000	29 976	9	605	1434	17 298	7	15028	1606	33	16 495
(4)	rand3000	4 928	9	554	21	1 918	6	8	823	1	30
	rand6000	10 293	11	836	95	5 598	9	25	1563	30	150
	rand10000	17 578	13	1192	376	10 706	10	381	2535	112	869
	rand15000	26 717	14	1570	875	17 810	12	354	3758	1540	2 769
	rand16000*	28 624	13	1612	826	19 700	13	2063	4002*	3028	5 917
	rand20000*	35 975	14	1993	1904	24 786	14	3632	4963*	8457	13 993
(5)	Gab100	182	7	3	<1	162	7	<1	24	1	1
	Gab300	552	10	5	<1	516	10	1	70	23	25
	Gab500	949	13	4	1	919	12	56	115	181	238
	Gab600*	1 174	14	11	2	1 097	14	5	135*	3067	3 074
	Gab700*	1 302	14	8	1	1 255	14	9	162*	5700	5 710
(6)	p1277	2 128	9	116	9	1 353	9	13	323	2	24
	p2518	4 266	9	329	32	1 876	5	27	621	1	60
	p4206	7 124	8	513	92	3 543	6	16	1057	2	110
	p5995	10 082	8	738	188	4 920	5	20	1495	1	209
	p7595	12 788	7	965	336	5 908	6	18	1903	<1	354

For instances of large size and small branchwidth, Step III may not dominant the running time. Our results show that the $O(n^3)$ time data reduction and branch decomposition finding take more time than the dynamic programming part for those instances.

Table 1 only contains the instances well scaled within some size ranges. We have tested FT Algorithm on graphs with size different from those in Table 1. The results are similar to those in the table, the running time mainly depends on $l(H)$ and then $|E(H)|$.

FT Algorithm requires in Step III $O(2^{10.1l(H)}k^2)$ and $O(2^{10.1l(H)}k)$ memory spaces for computing a minimum dominating set and $\gamma(H)$ of kernel H, respectively. The memory requirement seems a bottleneck for solving instances with large branchwidth. We report the memory space (in MBytes) used by FT Algorithm in Table 2 for large instances in Classes (1)–(6). Our data show that FT Algorithm can compute a minimum dominating set and the dominating number for instances with the branchwidth of kernels at most 13 ($l(H) \leq 13$) and at most 14 ($l(H) \leq 14$), respectively, by 3 GBytes memory space. The average memory space over three graphs used by FT Algorithm for some large instances is given in Table 3.

Our computational results confirm the theoretical analysis of FT Algorithm: It is efficient for graphs with small branchwidth but time and memory consuming for graphs with large branchwidth. This suggests that the branchwidth of a planar graph is a key parameter to decide if a problem on the graph can be solved efficiently or not. For example, Class (1) graphs have branchwidth at most four and thus admit efficient algorithms for many hard problems. On the other hand, the problems on graphs in Classes (2) and (5) are less tractable because these graphs have large branchwidth.

Both the theoretical analysis and computational study suggest that computing a kernel H with smaller $l(H)$ and $|E(H)|$ is a most effective way to improve the efficiency of FT Algorithm. For this purpose, we proposed new reduction rules (Rule 3). Recall that H is the kernel obtained by new reduction rules (Rules 1, 2, 3, and 4) and let H' be the kernel obtained by applying only the previous known reduction rules (Rules 1, 2, and 4). Since all nodes colored black (resp. nodes deleted) by previous rules are also colored (resp. deleted) by new rules, $l(H) \leq l(H')$ and $|E(H)| \leq |E(H')|$. For Classes (2) and (5), $l(H) = l(H') = bw(G)$ and $|E(H)| = |E(H')| = |E(G)|$ for most instances, that is, the effect of data reduction is very limited. However, for instances in other classes, data reduction is effective and our new rules improve the efficiency of FT Algorithm. For instances of Classes (1), (3), (4), and (6), Table 4 shows the computational results of FT Algorithm when previous rules and new rules are used. In the table, t_{old} and t_{new} (resp. $|B'|$ and $|B|$) are the total running times (resp. the numbers of vertices in an optimal dominating set decided in Step I) when previous rules and new rules are used, respectively. The data show that $l(H) = l(H')$ and $|E(H)| < |E(H')|$ for most instances. The total running time is improved when new rules are used:

Table 2
Memory space (in MBytes) of FT Algorithm with the index method in Step III for instances of Classes (1)–(6). For the instances marked with "*", the memory space is for computing the dominating number only. *X* indicates that the problem can not be solved for the instance because it requires more than 3 Gbytes memory space.

Class	Instance	$\|E(G)\|$	$bw(G)$	$l(H)$	Memory (MBytes)
(1)	max11000	28 537	4	4	480
	max13500	38 067	4	3	720
(2)	a280	788	13	13	510
	rd400*	1 183	17	17	*X* (>3000)
(3)	tri8000	23 975	9	7	710
	tri10000	29 976	9	7	1210 (Step III,1030)
(4)	rand10000	17 578	13	10	470
	rand15000	26 717	14	12	1800
	rand20000*	35 975	14	14	2000
	rand25000*	45 278	15	15	*X* (>3000)
(5)	Gab300	552	10	10	40
	Gab500	949	13	12	660
	Gab700*	1 302	14	14	1200
	Gab800*	1 533	16	16	*X* (>3000)
(6)	p5995	10 082	8	5	150
	p7595	12 788	7	6	240

Table 3
Average performance of FT Algorithm over three graphs for each instance size. The time is in seconds and memory is in Mbytes.

Class	Graph	Average $\|E(G)\|$	Average $bw(G)$	Average $l(H)$	Average $\gamma(G)$	Average time	Worst time	Average memory	Worst memory
(1)	max13500	38 322	4	3	1763	1 011	1 204	720	720
(3)	tri10000	29 973	9	7	1609	13 342	16 495	1200	1360
(4)	rand15000	26 706	15	11	3764	2 040	2 769	1190	1800
(5)	Gab500	945	13	12	117	200	238	670	710
(6)	p7595	12 760	7	6	1894	316	353	240	240

Table 4
The results (time in seconds) of using new data reduction rules and without using the new rules in Step I.

Class	Graph G	$\|E(G)\|$	$bw(G)$	Results without new rules				Results with new rules			
				$\|B'\|$	$\|E(H')\|$	$l(H')$	Time	$\|B\|$	$\|E(H)\|$	$l(H)$	Time
(1)	max1500	3 860	4	225	118	3	12	228	78	3	12
	max6000	7 480	4	2212	41	2	58	2214	32	2	55
	max8000	13 395	4	2183	218	3	353	2186	194	3	337
	max11000	28 537	4	1671	287	4	892	1679	208	4	800
	max13500	38 067	4	1752	362	4	1 291	1758	302	3	1 204
(3)	tri2000	5 977	8	102	3 787	7	353	136	3 192	7	198
	tri4000	11 969	9	214	7 541	7	1 941	252	6 888	7	1 903
	tri6000	17 979	9	277	12 370	8	3 895	312	11 691	8	3 576
	tri8000	23 975	9	421	14 953	7	10 192	497	13 524	7	7 750
	tri10000	29 976	9	551	18 273	7	18 945	605	17 298	7	16 495
(4)	rand3000	4 928	9	545	1 987	6	30	554	1 918	6	30
	rand6000	10 293	11	832	5 675	9	154	836	5 598	9	150
	rand10000	17 578	13	1176	10 861	11	892	1192	10 706	10	869
	rand13000	22 953	13	1454	14 856	10	1 662	1589	14 646	10	1 169
	rand15000	26 717	14	1553	17 984	12	2 834	1570	17 810	12	2 769
(6)	p1277	2 128	9	112	1 371	9	41	116	1 353	9	24
	p2518	4 266	9	291	2 139	6	69	329	1 876	5	60
	p4206	7 124	8	478	3 780	6	116	513	3 542	6	110
	p5995	10 082	8	671	5 372	5	224	738	4 920	5	209
	p7595	12 788	7	917	6 231	6	363	965	5 908	6	354

$t_{new} < t_{old}$ for most instances in the table. The improvement is instance dependent and t_{new}/t_{old} varies from 56% to 100%. The average of t_{new}/t_{old} over the five instances of Class (1) is about 95%. Similarly, the averages of t_{new}/t_{old} for Classes (3), (4), and (6) are about 80%, 90%, and 85%, respectively. The improvement of the total running time is obtained mainly from Step III. The running time of Step I when new rules are used is about the same as that when previous rules are used (instance dependent) and we omit the details here.

Table 5
The results (time in seconds) of using distance product method and index method in Step III.

Class	Graph	$\|E(G)\|$	Distance product time	Index method time
(1)	max8000	13 395	<1	<1
	max11000	28 539	2	<1
	max13500	38 067	<1	<1
(2)	pr299	864	25	35
	tsp225	622	104	109
	a280	778	310	336
(3)	tri5000	14 969	56	7
	tri6000	17 974	62	11
	tri7000	20 980	163	14
(4)	rand5000	8 451	12	2
	rand6000	10 293	21	30
	rand8000	13 816	83	38
(5)	Gab100	182	1	1
	Gab200	366	1	2
	Gab300	552	18	23
(6)	p1277	2 128	3	2
	p5995	10 092	46	3
	p7595	12 691	5	1

Step III of FT Algorithm can also be realized by the distance product method proposed by Dorn [15]. When a conventional $O(n^3)$ time method is used to realize the distance product of matrices, the distance product method has the same time complexity as that of the index method. Theoretically, using the fast matrix multiplication for the distance product of integer matrices [44] can reduce the order of time complexity. In practice, using the fast matrix multiplication (e.g., the Strassen's method) for distance product of matrices is slower than the conventional method. We report in Table 5 the running times of Step III by the index method and the distance product method (with conventional distance product). Our data show that the running times of the two methods are similar. Both methods require a similar size of memory space as well.

5. Concluding remarks

We tested the practical performances of FT Algorithm on a wide range of planar graphs. The computational results coincide with the theoretical analysis of the algorithm, it is efficient for graphs with small branchwidth but may not be practical for graphs with large branchwidth. By a computer with a CPU of about 2.4 GHz and 3 GMBytes memory space, it is possible to find a minimum dominating set (resp. the dominating number) for graphs with the branchwidth of their kernels at most 13 (resp. 14) in a few hours. Since FT Algorithm runs and requires memory space exponentially in the branchwidth $l(H)$ of a kernel H for a given graph, it is worth to develop more powerful data reduction rules to reduce $l(H)$. It is also worth to develop heuristics to reduce $l(H)$ to compute approximate solutions for the PLANAR DOMINATING SET problem by branch-decomposition based algorithms. Those heuristics should provide guaranteed solutions very close to the optima but run faster and use less memory space than FT Algorithm for graphs with large branchwidth. It is interesting to compare the performance of FT Algorithm with the PTAS for the PLANAR DOMINATING SET problem. Another research problem is to perform computational studies on the subexponential algorithms for other problems, especially those non-local problems such as the longest path and connected dominating set problems, in planar graphs.

Acknowledgements

The authors thank anonymous reviewers for constructive comments. The work was partially supported by NSERC Research Grant of Canada and Research Grant Council of Hong Kong (Project No. CityU 114307).

Appendix

We show that the instances of Class (1) (the random maximal graphs and their subgraphs generated by LEDA) has branchwidth at most four. We prove this statement by constructing a branch decomposition of width at most four for any maximal graph in this Class. Let G_n be a maximal graph of n vertices in Class (1). For $n = 3$, G_n is the graph with three edges (see Fig. 1). For $n \geq 4$, G_n is created by adding a new vertex in a randomly chosen face f of G_{n-1} and three edges between the new vertex and the three vertices incident to f [2,34]. Let $E_3 = E(G_3)$. For $4 \leq j \leq n$, let u_j and E_j be the new vertex and the set of edges, respectively, added to create G_j. For each E_i, $3 \leq i \leq n$, we create a rooted binary tree T_i with root r_i and three leaves (see Fig. 1). We assign the three edges of E_i to the leaves of T_i, one edge per leaf in an arbitrary way. We say E_i is a *parent* of E_j if $i < j$ and the end vertices of the three edges of E_j except u_j are also end vertices of edges of E_i. If E_i is a

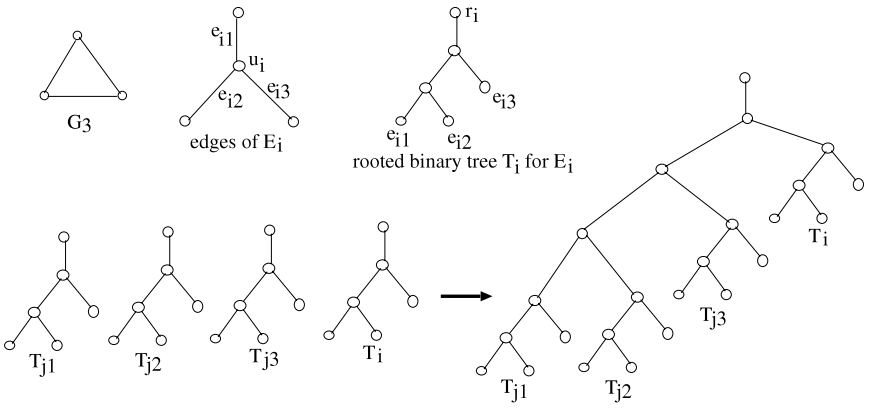

Fig. 1. Merge the rooted binary trees T_i into one binary tree.

parent of E_j, E_j is called a child of E_i. Obviously for $j \geq 4$, each E_j has a unique parent and for $i \geq 3$, each E_i has at most three children. We merge the rooted binary trees T_i, $3 \leq i \leq n$, into a rooted binary tree T by the following recursive procedure.

- Merge_Tree(E_i)
  If E_i has any child E_j then call Merge_Tree(E_j) for every child E_j of E_i; otherwise output the rooted binary tree T_i and RETURN.
  Merge the rooted trees obtained from Merge_Tree(E_j) for all E_j by connecting the roots of the binary trees into one rooted binary tree T'; merge T' and T_i by connecting the roots of them into a rooted binary tree T (see Fig. 1 for the merge process); output T and RETURN.

Calling Mereg_Tree(E_3) merges all rooted binary trees T_i, $3 \leq i \leq n$, into a rooted binary tree T. For an arbitrary link e of T, let T_e be the subtree consisting of all descendant links of e in T. For a link e of T such that T_e has at most two leaves, $|S_e| \leq 4$. For a link e such that T_e has at least three leaves, T_e has at least one rooted binary tree T_j as a subtree. For such a link e, an E_j is maximal in T_e if T_e does not have a subtree T_i for E_i such that E_i is a parent of E_j. If T_e has only one maximal E_j then $S_e \subseteq V(E_j)$, where $V(E_j)$ is the set of end vertices of edges in E_j. Assume that T_e has more than one maximal E_j. Let E_i be the parent of those E_j's. Then $S_e \subseteq V(E_i)$. In either cases, $|S_e| \leq 4$. Therefore, T is a branch decomposition of G_n (we need to remove the root of T) with width at most four. This implies that G_n has branchwidth at most four. It is known that the branchwidth of a subgraph of G_n is at most the branchwidth of G_n. Thus, the instances of Class (1) have branchwidth at most four.

References

[1] Public Implementation of a Graph Algorithm Library and Editor. http://pigale.sourceforge.net/, 2008.
[2] The LEDA User Manual, Algorithmic Solutions, Version 4.2.1. http://www.mpi-inf.mpg.de/LEDA/MANUAL/MANUAL.html, 2008.
[3] J. Alber, N. Betzler, R. Niedermeier, Experiments on data reduction for optimal domination in networks, in: Proc. of the International Network Optimization Conference, INOC2003, 2003, pp. 1–6.
[4] J. Alber, H.L. Bodlaender, H. Fernau, T. Kloks, R. Niedermeier, Fixed parameter algorithms for dominating set and related problems on planar graphs, Algorithmca 33 (2002) 461–493.
[5] J. Alber, F. Dorn, R. Niedermeier, Experiments on optimally soving NP-complete problems on planar graphs. Manuscript, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.23.4973\&rep=rep1\&type=pdf.
[6] J. Alber, F. Dorn, R. Niedermeier, Experimental evaluation of a tree decomposition-based algorithm for vertex cover on planar graphs, Discrete Applied Mathematics 145 (2) (2005) 219–231.
[7] J. Alber, H. Fan, M. Fellows, H. Fernau, R. Niedermeier, Refined search tree technique for dominating set on planar graphs, in: Proc. of the 26th Mathematical Foundations of Computer Science, MFCS2001, in: LNCS, vol. 2136, 2001, pp. 111–122.
[8] J. Alber, M.R. Fellows, R. Niedermeier, Polynomial time data reduction for dominating set, Journal of the ACM 51 (3) (2004) 363–384.
[9] B.S. Baker, Approximation algorithms for NP-complete problems on planar graphs, Journal of ACM 41 (1994) 153–180.
[10] Z. Bian, Q. Gu, Computing branch decompositions of large planar graphs, in: Proc. of the 7th International Workshop on Experimental Algorithms, WEA 2008, in: LNCS, vol. 5038, 2008, pp. 87–100.
[11] Z. Bian, Q. Gu, M. Marzban, H. Tamaki, Y. Yoshitake, Empirical study on branchwidth and branch decomposition of planar graphs, in: Proc. of the 9th SIAM Workshop on Algorithm Engineering and Experiments, ALENEX'08, 2008, pp. 152–165.
[12] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, Journal of Symbolic Computation 9 (1990) 251–280.
[13] E.D. Demaine, F.V. Fomin, M.T. Hajiaghayi, D.M. Thilikos, Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs, Journal of ACM 52 (6) (2005) 866–893.
[14] E.D. Demaine, M.T. Hajiaghayi, The bidimensionality thoery and its algorithmic applications, Computer Journal 51 (3) (2008) 292–302.
[15] F. Dorn, Dynamic programming and fast matrix multiplication, in: Proc. of the 14th Annual European Symposium on Algorithms, ESA2006, in: LNCS, vol. 4168, 2006, pp. 280–291.
[16] F. Dorn, How to use planarity efficiently: New tree-decomposition based algorithms, in: Proc. of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science, WG2007, in: LNCS, vol. 4769, 2007, pp. 280–291.
[17] F. Dorn, F.V. Fomin, D.M. Thilikos, Catalan structures and dynamic programming in H-minor-free graphs, in: Proc. of 2008 ACM/SIAM Symposium on Discrete Algorithms, SODA08, 2008, pp. 631–640.
[18] F. Dorn, F.V. Fomin, D.M. Thilikos, Subexponential parameterized algorithms, Computer Science Review 2 (1) (2008) 29–39.

[19] F. Dorn, E. Penninkx, H. Bodlaender, F.V. Fomin, Efficient exact algorithms for planar graphs: Exploiting sphere cut branch decompositions, in: Proc. of the 13th Annual European Symposium on Algorithms, ESA05, in: LNCS, vol. 3669, 2005, pp. 95–106.
[20] R.G. Downey, M.R. Fellow, Parameterized complexity, in: Monographs in Computer Science, Springer-Verlag, 1999.
[21] R.G. Downey, M.R. Fellows, Fixed parameter tractability and completeness, Congressus Numerantium 87 (1992) 161–187.
[22] U. Fiege, A threshold of ln n for approximating set cover, Journal of ACM 45 (1998) 634–652.
[23] F.V. Fomin, D.M. Thilikos, Dominating sets in planar graphs: Branch-width and exponential speed-up, SIAM Journal on Computing 36 (2) (2006) 281–309.
[24] F.V. Fomin, D.M. Thilikos, New upper bounds on the decomposability of planar graphs, Journal of Graph Theory 51 (1) (2006) 53–81.
[25] M.R. Garey, D.S. Johnson, Computers and Intractability, a Guide to the Theory of NP-Completeness, Freeman, New York, 1979.
[26] Q. Gu, H. Tamaki, Optimal branch-decomposition of planar graphs in $O(n^3)$ time, ACM Transactions on Algorithms 4 (3) (2008) 30:1–30:13.
[27] T.W. Haynes, S.M. Hedetniemi, S.T. Hedetniemi, M.A. Henning, Domination in graphs applied to electronic power networks, SIAM Journal on Discrete Mathematics 15 (4) (2002) 519–529.
[28] T.W. Haynes, S.T. Hedetniemi, P.J. Slater, Domination in graphs, in: Monographs and Textbooks in Pure and Applied Mathematics, vol. 209, Marcel Dekker, 1998.
[29] T.W. Haynes, S.T. Hedetniemi, P.J. Slater, Fundamentals of domination in graphs, in: Monographs and Textbooks in Pure and Applied Mathematics, vol. 208, Marcel Dekker, 1998.
[30] D.S. Johnson, Approximation algorithms for combinatorial problems, Journal of Computer and System Sciences 9 (1974) 256–278.
[31] I.A. Kanj, L. Perkovic, Improved parameterized algorithms for planar dominating set, in: Proc. of the 27th MFCS, in: LNCS, vol. 2420, 2002, pp. 399–410.
[32] Xiang-Yang Li, Algorithmic, geometric and graphs issues in wireless networks, Journal of Wireless Communications and Mobile Computing (WCMC) 6 (2) (2003) 119–140.
[33] M. Marzban, Q. Gu, X. Jia, Computational study on dominating set problem of planar graphs, in: Proc. of the 2nd International Conference on Combinatorial Optimization and Applications, COCOA 2008, in: LNCS, vol. 5165, 2008, pp. 89–102.
[34] K. Mehlhorn, S. Näher, LEDA: A Platform for Combinatorial and Geometric Computing, Cambridge University Press, New York, 1999.
[35] G. Reinelt, TSPLIB—A traveling salesman library, ORSA Journal on Computing 3 (1991) 376–384.
[36] N. Robertson, P.D. Seymour, Graph minors I. Excluding a forest, Journal of Combinatorial Theory, Series B 35 (1983) 39–61.
[37] N. Robertson, P.D. Seymour, Graph minors II. Algorithmic aspects of tree-width, Journal of Algorithms 7 (1986) 309–322.
[38] N. Robertson, P.D. Seymour, Graph minors X. Obstructions to tree decomposition, Journal of Combinatorial Theory, Series B 52 (1991) 153–190.
[39] L.A. Sanchis, Experimental analysis of heuristic algorithms for the dominating set problem, Algorithmica 33 (2002) 3–18.
[40] G. Schaeffer, Random sampling of large planar maps and convex polyhedra, in: Proc. of the 31st Annual ACM Symposium on the Theory of Computing, STOC'99, 1999, pp. 760–769.
[41] P.D. Seymour, R. Thomas, Call routing and the ratcatcher, Combinatorica 14 (2) (1994) 217–241.
[42] P.J. Wan, K.M. Alzoubi, O. Frieder, A simple heuristic for minimum connected dominating set in graphs, International Journal of Foundations on Computer Science 14 (2) (2003) 323–333.
[43] D.B. West, Introduction to Graph Theory, Prentice Hall Inc., Upper Saddle River, NJ, 1996.
[44] U. Zwick, All pairs shortest paths using bridging sets and rectangular matrix multiplication, Journal of ACM 49 (2002) 289–317.