A note on Baker's algorithm

Iyad A. Kanj, Ljubomir Perković

School of CTI, DePaul University,
243 S. Wabash Avenue, Chicago, IL 60604-2301.

Abstract

We present a corrected version of Baker's algorithm for finding a minimum dominating set in an l-outerplanar graph.

Key words: Graph algorithms, outerplanar graphs, dominating set

1 Introduction

Baker, in her seminal work [3], gave a general technique to devise approximation schemes for the maximum independent set problem on planar graphs. Her technique is partially based on a dynamic programming algorithm that, for a planar graph of outerplanarity l, computes a maximum independent set of the graph in time $O(8^l n)$, where n is the number of vertices in the graph [3]. Baker also mentions in her paper that the $O(8^l n)$ algorithm, with a slight modification and without any increase in running time, can be applied to solve the minimum dominating set problem on planar graphs ([3], pages 175 and 176-177). If Baker's algorithm works, then her algorithm, together with the fact that a planar graph with a dominating set of size bounded by k can be at most $3k$-outerplanar, would imply that the Dominating Set problem on planar graphs is solvable in time $O(8^{3k} n)$, as observed in [1]. Many recent papers, in fact, cite and/or make use of Baker's algorithm for the dominating set problem (e.g. [1], [2], [4], [5], [6], [7]).

However, Baker's algorithm in its current form cannot be applied to solve the minimum dominating set problem. This is essentially because the dominating

Email addresses: `ikanj@cs.depaul.edu` (Iyad A. Kanj),
`lperkovic@cs.depaul.edu` (Ljubomir Perković).

set problem, unlike the independent set problem, does not exhibit an optimal substructure property as we argue in section 2.

Baker's algorithm *can* be modified to solve the minimum dominating set problem, albeit with an increase in the running time to $O(27^l n)$. Because Baker's paper focuses on the independent set problem and it is a non-trivial task to translate Baker's algorithm to the dominating set problem, and because of the recent interest in Baker's algorithm for the dominating set problem [1,2,4,7], we provide a brief but complete description of the corrected Baker's algorithm for computing a minimum dominating set in an l-outerplanar graph.

2 The issue with Baker's algorithm

Let O be a maximum independent set in a graph $G = (V, E)$, and let S be a separator of G. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be the subgraphs of G obtained by removing S, and let G_1' and G_2' be the subgraphs of G induced by $V_1' = V_1 \cup S$ and $V_2' = V_2 \cup S$, respectively. Let $I_1 = O \cap V_1'$ and $I_2 = O \cap V_2'$. Then I_1 and I_2 are maximum independent sets of G_1' and G_2', respectively, subject to the constraint that they agree with O on the vertices in S. Thus, a maximum independent set of G can be computed by combining some two maximum independent sets I_1 and I_2 of graphs G_1' and G_2', respectively, that agree on S. The pair I_1, I_2 can then be found by listing, for every subset X of S, maximum independent sets of G_1' and G_2' whose intersection with S is exactly X. Baker, in her dynamic programming algorithm, uses this insight to construct the maximum independent set of G as follows. Each vertex in some separator S of G is assigned a state: 1 (in the independent set) or 0 (not in the independent set). The algorithm is then called on G_1' and G_2' recursively subject to the constraints on the vertices in S. The maximum independent set is found by trying every possible assignment of states to vertices in S.

Now, consider O to be a minimum dominating set in $G = (V, E)$ and let S, G_1, G_2, G_1' and G_2' be defined as above. It is generally not true that $O \cap V_1'$ is the minimum among all dominating sets of G_1' that agree with O on the vertices in S (in fact $O \cap V_1'$ may not even be a dominating set for G_1', e.g. $V = \{1, 2, 3, 4\}$, $E = \{(1, 2)(2, 3)(3, 4)\}$, $S = \{1, 3\}$ and $O = \{1, 4\}$). Thus, one cannot in general construct a minimum dominating set for G by combining minimum dominating sets D_1 and D_2 of graphs G_1' and G_2' that agree on the vertices in S. The problem is that if a vertex in the separator S is not in the minimum dominating set, then it could be dominated by a vertex in V_1 or by a vertex in V_2. Two states (in the dominating set, or out of the dominating set) for each vertex of the separator are not enough. For this reason, we must consider 3 states for each vertex in the separator: 1 if the vertex is in the dominating set, 0 if the vertex is not in and must be dominated, and 2 if the vertex is not in and does not need to be dominated (because it will presumably be dominated by a vertex on the "other side" of the separator).

As we discuss in the remainder of this paper, Baker's algorithm *can* be used for the minimum dominating set problem, albeit with 3 states instead of 2 which changes the algorithm's running time to $O(27^l n)$. We start with some definitions.

3 Definitions

A graph is called **outerplanar (or 1-outerplanar)** if it has an embedding in the plane such that every vertex lies on the unbounded face. Given a connected outerplanar graph $G = (V, E)$ in such an embedding, we can construct a corresponding **outerplanar decomposition tree** $\overline{G}$. (We assume, as Baker [3] did, that the planar embedding is given to us by an appropriate data structure such as that used by Lipton and Tarjan [8].) In order to construct the tree, we first prepare the graph by adding a duplicate edge to every bridge of G (so a face is created between two cutpoints). We then construct a rooted tree $\overline{G}$ whose vertices correspond to the exterior edges and the interior faces of G. Each vertex corresponding to an exterior edge (x, y) is a leaf of $\overline{G}$ labeled (x, y) and is connected to the vertex of $\overline{G}$ corresponding to the face of G containing (x, y). Pairs of vertices of $\overline{G}$ corresponding to neighboring (i.e. sharing an edge) faces of G are also connected and one such vertex is designated as the root. (If G has cutpoints then we actually obtain a forest. To make it into a tree, we repeatedly choose disjoint trees T_1 and T_2 each containing an internal node corresponding to a face incident to a particular cutpoint, and we add an edge between these nodes.) Once the root is defined, we label each internal vertex v of the tree, starting from the bottom of the tree, as follows: we order the children of v so that their labels (i.e. corresponding edges) are listed counterclockwise, say (x, u_1), $(u_1, u_2), \ldots, (u_l, y)$, and then we assign v label (x, y). Note that if $x \neq y$, (x, y) is the edge between the face of G corresponding to v and the face corresponding to v's parent in $\overline{G}$; if $x = y$ then x is a cutpoint, unless it is the root of $\overline{G}$.

A node of a planar graph G in a planar embedding is said to be at level 1 if it is on the exterior face. Let L_1 be the subset of level 1 vertices. We define L_i to be the set of all nodes on the exterior face of the graph G after levels $L_1, \ldots, L_{i-1}$ have been removed. A graph is called **l-outerplanar** if $L_l \neq \emptyset$ and $L_{l+1} = \emptyset$.

Now, given a connected l-outerplanar graph G, we construct one outerplanar decomposition tree for every level-i connected component of G and every $i = 1, 2, ..., l$. More precisely, there will be a tree for the level 1 component, and also one for every level $i > 1$ connected component inside a level $i-1$ face; if there are disconnected level i components inside a level $i-1$ face, we add bridges – and duplicate them as described above – to connect them (but we ignore them in the dynamic programming algorithm). Thus each level $i > 1$ component C inside a level $i-1$ face is an outerplanar graph and has an outerplanar tree decomposition $\overline{C}$. In order to label the tree vertices, we construct a planar triangulation of G (done *after* the addition of bridges) so that every added edge is between vertices in different levels. We then label the tree vertices, starting with the labeling of level 1 tree vertices as described above. Assuming inductively a labeling of the level $i-1$ tree vertices, we label the level $i > 1$ tree vertices as usual, after choosing the root of each tree and its leftmost child as follows. Consider a level $i > 1$ component C enclosed inside a level $i-1$ face with corresponding level $i-1$ tree vertex already labeled (x, y) (where $x = y$ is possible). The root of $\overline{C}$ will be the tree node corresponding to the face of C containing the node of C (say, z) adjacent to both x and y in the triangulation (which is unique except if $x = y$ in which case we arbitrarily choose a neighbor of x). We label the root (z, z) and set its leftmost child to be the tree vertex corresponding to the exterior edge of C incident to and counterclockwise from z. In what follows, we will often identify a tree vertex with its labeling. We note that the total number of tree vertices is $O(|E(G)|) = O(n)$ because each tree leaf corresponds to a unique edge of G and each internal tree vertex has at least two children.

We now match each vertex of each tree to a subgraph of G we call a **slice**. In order to define slices precisely, we introduce some terminology. Let C be a level $i > 1$ component enclosed inside a level $i-1$ face f. Let (x_1, x_2) and (x_2, x_3) be two exterior edges of C listed in counterclockwise order. A vertex of f is called a **dividing point** for the ordered edge pair $((x_1, x_2), (x_2, x_3))$ if it is adjacent to x_2 in the triangulation and appears before x_3 when listing the neighbors of x_2 in a counterclockwise manner, starting from x_1. Note that, in the planar triangulation of G, every pair of adjacent exterior edges (ordered counterclockwise) of every level $i > 1$ component will have at least one dividing point.

We now define, for every tree vertex (x, y), the left and right **boundaries**, which are two sets of vertices that together form a separator in G that bounds the slice corresponding to (x, y). If (x, y) is a level 1 leaf, then its left boundary is $\{x\}$, its right boundary is $\{y\}$ and its slice is just the edge (x, y). If (x, y) is a level i face, with leftmost child (x, w) and rightmost child (z, y), then the left (resp. right) boundary of (x, y) is the left (resp. right) boundary of (x, w) (resp. (z, y)). As shown by Baker [3], the slice for (x, y) is (x, y) (if it is an edge) together with either: (a) the union of the slices of the children of (x, y) if face (x, y) encloses no level $i+1$ components, or (b), if (x, y) encloses a level $i+1$ component C, the slice corresponding to the root of $\overline{C}$.

Finally, we define boundaries of leaves of each of level $i > 1$ tree $\overline{C}$ as follows. Let (x_1, x_1) be the root of $\overline{C}$ and let $(x_1, x_2), ..., (x_i, x_{i+1}), ..., (x_t, x_{t+1})$, where $x_{t+1} = x_1$, be the exterior edges of C (and also the leaves of $\overline{C}$), listed in counterclockwise order. Let C be enclosed by a level $i-1$ face f labeled (y_1, y_l) (where y_1 and y_l could be equal), and let the level $i-1$ tree vertex corresponding to f have children $(y_1, y_2), ..., (y_{l-1}, y_l)$, listed in order from left to right. Then the left boundary of (x_1, x_2) is x_1 together with the left boundary of (y_1, y_2), and the right boundary of (x_t, x_1) is x_1 together with the right boundary of y_{l-1}, y_l. For $1 < i \leq t$, the left boundary of (x_i, x_{i+1}) is x_i together with the left boundary of (y_j, y_{j+1}) where y_j is the first (when vertices of f are listed in counterclockwise order starting with y_1) dividing point for (x_{i-1}, x_i) and (x_i, x_{i+1}). For $1 \leq i < t$, the right boundary of (x_i, x_{i+1}) will be equal to the left boundary of (x_{i+1}, x_{i+2}). The slice for any (x_i, x_{i+1}) whose left and right boundaries differ will consist of the union of slices of $(y_r, y_{r+1}), ..., (y_{s-1}, y_s)$ of face f, where y_r is in the left boundary and y_s is in the right boundary of (x_i, x_{i+1}), together with edges between x_i, x_{i+1}, y_r, ..., and y_s. If the boundaries are the same, then the slice is (x_i, x_{i+1}) together with the boundary. Note that for every pair of sibling tree vertices (x, y) and (y, z) that are adjacent in the sibling ordering, the right boundary of (x, y) will be the same as the left boundary of (y, z).

4 The algorithm

We now describe the details of Baker's algorithm for computing a minimum dominating set in an l-outerplanar graphs. Baker's algorithm computes a table for every vertex of every level i tree ($1 \leq i \leq l$), starting at the leaves of the level 1 tree. Each table T will contain the size $m_T(s)$ (or simply $m(s)$) of the minimum dominating set in the corresponding slice for every possible state s of the boundary, i.e. for every assignment of states 1 (in the dominating set), 0 (not in the dominating set and must be dominated by a vertex in the slice), and 2 (not in the dominating set and does not need to be dominated) to the vertices in the boundary. So, if the boundary has q vertices, there will be 3^q table entries. When a table has been computed for every vertex of every level i tree ($1 \leq i \leq l$), the size of the minimum dominating set for G can be read out from the table associated with the root of the level 1 tree, labeled, say, (x, x), as follows: it will be the minimum of the two entries in the table corresponding to states $(0, 0)$ (graph node x not in the dominating set) and $(1, 1,)$ (x in the dominating set).

(x, y) is a level 1 leaf

If tree node (x, y) is a level 1 leaf then the corresponding slice and boundary consist of just $\{x, y\}$ and the following table is created:

state of x	0	0	0	1	1	1	2	2	2
state of y	0	1	2	0	1	2	0	1	2
$m(s)$	∞	1	∞	1	2	1	∞	1	0

Clearly, creating the above table takes $\Theta(1)$ time, and is done no more than $O(n)$ times.

(x, y) is a level i non-leaf: case one

If (x, y) is a level i internal tree node corresponding to a face that **does not contain a level $i + 1$ component**, then the table T for (x, y) is computed by successively merging the tables of the children $(x, z_1), (z_1, z_2), \ldots, (z_l, y)$ of (x, y) and then adjusting the final table. We describe below the procedures Merge and and Adjust.

Merge takes two level i slices (x, z) and (z, y) such that the right boundary of (x, z) is the same as the left boundary of (z, y). A new table T_0 is created with 3^{2i} entries, corresponding to a new slice whose left boundary is the left boundary of (x, z) and whose right boundary is the right boundary of (z, y). For every state s of the boundary of the new slice (i.e. an assignment of states $0, 1$ and 2 to vertices in the boundary), there corresponds a set S_x of 3^i boundary states in the table T_1 of (x, z) that agree with s on the left boundary, and a set S_y of 3^i boundary states in the table T_2 of (z, y) that agree with s on the right boundary. We say that a boundary state s_1 in S_x **matches** with a boundary state s_2 in S_y if the following is true: for every vertex v in the right boundary of (x, z) (i.e. the left boundary of (z, y)) either the state for v is 1 in both s_1 and s_2, or it is 0 in one and 2 in the other. We consider all 3^i pairs of matching boundary states s_1 and s_2 in S_x and S_y, respectively, and for each pair we add the values $m_{T_1}(s_1)$ and $m_{T_2}(s_2)$, and then we subtract from this sum the number of vertices in the boundary shared by (x, z) and (z, y) assigned 1 in s_1 (and s_2). We set $m_{T_0}(s)$ to be the minimum, over all 3^i matching states s_1 and s_2, of this value. The running time of **Merge** is $O(3^{3i})$.

Adjust: If $x = y$ (i.e. x is a cut vertex of its level-i component C, or (x, x) is the root of $\overline{C}$) then we set $m_T(s) = \infty$ for every boundary state s with different states for x and y. In addition, we decrement $m_T(s)$ by 1 for every boundary state s which assigns 1 to both x and y. If $x \neq y$ and (x, y) is an edge, for every boundary state s in which x and y are assigned 1 and 0, respectively, we set $m_T(s) = m_T(s_2)$ where s_2 is a boundary state that agrees with s on every vertex except y, which is assigned 2 instead. We repeat this for assignments in which x and y are assigned 0 and 1, respectively. **Adjust** runs in $O(3^{2i})$ time and is called at most once for each tree vertex.

(x, y) is a level i non-leaf: case two

If (x, y) is a level i tree vertex corresponding to a face f that contains a level $i+1$ component C, then the algorithm is recursively called on $\overline{C}$ and a table T' for its root, labeled, say, (z, z), is computed. Note that the left (resp. right) boundary of (z, z) is z together with the left (resp. right) boundary of (x, y). Therefore, for every boundary state s in the table T for (x, y) there will correspond 3 boundary states s_0, s_1 and s_2 in T', one for each assignment of states $0, 1$ and 2 to z, respectively, and that agree with s on all vertices in the boundary of (x, y). We set $m_T(s)$ to $\min\{m_{T'}(s_0), m_{T'}(s_1), m_{T'}(s_2)\}$ if x is assigned 1 and there is an edge (x, z) in G or if y is assigned 1 and there is an edge (y, z) in G; otherwise, we set $m_T(s)$ to $\min\{m_{T'}(s_0), m_{T'}(s_1)\}$. We then run the Adjust procedure. Computing the whole table thus takes $\Theta(3^{2i})$ time, excluding the time taken by the recursive call, and this case will happen $O(n)$ times.

(x, y) is a level $i > 1$ leaf

If (x, y) is a level $i > 1$ leaf and the corresponding graph edge (x, y) is contained in a level $i-1$ face f, let $(x_1, x_2), (x_2, x_3), \ldots, (x_{l-1}, x_l)$ be the subset of children of the level $i-1$ tree vertex corresponding to f, listed from left to right, such that x_1 and x_l belong to the left and right boundaries of (x, y), respectively. We start by extending the table T (i.e. slice) of each level $i-1$ tree vertex (x_j, x_{j+1}) to include $z = x$ or $z = y$, depending on which forms a triangle with (x_j, x_{j+1}) in the planar triangulation of G. Then, each state s in the original table T will correspond to 3 new states s_0, s_1, and s_2 in the extended table T', one for each possible state 0, 1, and 2, respectively, for z. We set $m_{T'}(s_2) = m_T(s)$ and

$m_{T'}(s_0) = m_T(s)$ if x_i has state 1 in s_0 and $(x, x_i) \in E(G)$ or x_{i+1} has state 1 in s_0 and $(x, x_{i+1}) \in E(G)$; otherwise $m_{T'}(s_0) = \infty$.

$m_{T'}(s_1) = m_T(s)+1$ unless (a) x_i is assigned state 0 in s_1 and $(x, x_i) \in E(G)$, or (b) x_{i+1} is assigned state 0 in s_1 and $(x, x_{i+1}) \in E(G)$. If (a) holds but not (b), let s^* be the boundary state equal to s except that x_i is assigned 2. If (b) holds but not (a), let s^* be the boundary state equal to s except that x_{i+1} is assigned 2. If both (a) and (b) hold, let s^* be the boundary state equal to s except that x_i and x_{i+1} are both assigned 2. Then, $m_{T'}(s_1) = m_T(s^*) + 1$.

Each extension takes $\Theta(3^{2i})$ time and is done no more than $O(n)$ times. Next, we successively merge the tables of neighboring extended slices containing $z = x$ and, separately, of neighboring extended slices containing $z = y$. For every state s in the table T of a slice obtained by merging left slice S_1 with right slice S_2, we compute $m_T(s)$ as follows. If z is assigned 1 or 2 in s, we define the shared boundary to consist of the right boundary of S_1 (i.e. the left boundary of S_2) with z removed, and we compute $m_T(s)$ on this shared boundary as we did in the merge procedure. If z is assigned 0 in s, we define the shared boundary to consist of the right boundary of S_1 (i.e. the left boundary of S_2), including z, and we again compute $m_T(s)$ on this shared boundary as we did in the merge procedure except that states assigning 1 to z are not considered. Finally, we merge the resulting two slices, one containing x and the other containing y to obtain a table for (x, y), and call adjust on the resulting table to obtain the final table for (x, y).

We note that the total running time of the algorithm is bounded by the running time of **Merge** which takes $O(3^{3l}n)$ time in the worst case and is called $O(n)$ times. We conclude that the MINIMUM DOMINATING SET problem on l-outerplanar graphs on n vertices can be solved using this corrected Baker's algorithm in time $O(27^l n)$.

References

[1] J. ALBER, H.L. BODLAENDER, H. FERNAU, T. KLOKS AND R. NIEDERMEIER, Fixed parameter algorithms DOMINATING SET and related problems on planar graphs. *Algorithmica* **33** (2002), pp. 461-493.

[2] J. ALBER, H. FERNAU, AND R. NIEDERMEIER, Parameterized Complexity: Exponential Speed-Up for Planar Graph Problems. *Journal of Algorithms* **52** (2004), pp. 26-56.

[3] B. S. BAKER, Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM* **41** (1994), pp. 153-180.

[4] H. FERNAU AND D. JUEDES, A Geometric Approach to Parameterized Algorithms for Domination Problems on Planar Graphs. In *Proc. 29th International Symposium on Mathematical Foundations of Computer Science* (MFCS'04), *Lecture Notes in Computer Science* **3153**, (2004), pp. 488-499.

[5] F.V. FOMIN AND D.M. THILIKOS, Dominating sets in planar graphs: branch-width and exponential speed-up. In *Proc. 14th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA)* (2003), pp. 168-177.

[6] F.V. FOMIN AND D.M. THILIKOS, A simple and Fast Approach for Solving Problems on Planar Graphs. In *Proc. 21st International Symposium on Theoretical Aspects of Computer Science (TACS), Lecture Notes in Computer Science* **vol. 2996**, Springer (2004).

[7] I. KANJ AND L. PERKOVIĆ, Improved parameterized algorithms for planar dominating set. In *Proc. 27th International Symposium Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Comput. Sci.* **vol. 2420**, Springer (2002), pp. 399-410.

[8] R. LIPTON AND R. TARJAN, A Separator Theorem for Planar Graphs. *SIAM Journal of Applied Mathematics* **36** (1979), pp 177-189.