

# Connected odd dominating sets in graphs

Yair Caro

Department of Mathematics  
University of Haifa - Oranim  
Tivon - 36006, ISRAEL  
E-mail: yairc@macam.ac.il

William F. Klostermeyer  
Dept. of Computer and Information Sciences  
University of North Florida  
Jacksonville, FL 32224, U.S.A.  
E-mail: klostermeyer@hotmail.com

Raphael Yuster  
Department of Mathematics  
University of Haifa - Oranim  
Tivon - 36006, ISRAEL  
E-mail: raphy@research.haifa.ac.il

## Abstract

An *odd dominating set* of a simple, undirected graph  $G = (V, E)$  is a set of vertices  $D \subseteq V$  such that  $|N[v] \cap D| \equiv 1 \pmod{2}$  for all vertices  $v \in V$ . It is known that every graph has an odd dominating set. In this paper we consider the concept of connected odd dominating sets. We prove that the problem of deciding if a graph has a connected odd dominating set is NP-complete. We also determine the existence or non-existence of such sets in several classes of graphs. Among other results, we prove there are only 15 grid graphs that have a connected odd dominating set.

AMS subject classification index: primary 05C35.

Key Words: Dominating set, Odd dominating set.

# 1 Introduction

An *odd dominating set* of a simple, undirected graph  $G = (V, E)$  is a set of vertices  $D \subseteq V$  such that  $|N[v] \cap D| \equiv 1 \pmod{2}$  for all vertices  $v \in V$ , where  $N[v]$  denotes the closed neighborhood of  $v$ . Odd dominating sets and the analogously defined *even dominating sets* have received considerable attention in the literature, see [1, 3, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14]. Sutner proved that every graph contains at least one odd dominating set [14] and other proofs of this can be found in [4, 8]. Sutner also showed that deciding if a graph contains an odd dominating set of size at most  $k$  is NP-complete [14]. Caro et al. considered the size of the smallest odd dominating set in certain classes of graphs [5, 6].

One of the most important variations on the concept of a dominating set is that of a *connected dominating set* (cf. [13]). In this paper, we extend this notion to odd dominating sets by examining connected odd dominating sets or *CODS*. A CODS of a graph is an odd dominating set  $D$  such that the subgraph induced by  $D$  is connected. If  $D$  is a CODS, then either  $D$  is a single vertex or the subgraph induced by  $D$  is Eulerian. Thus, stars are the only trees that have CODS and, more generally, graphs having “real bridges” (bridges whose endpoints have degree at least 2) do not have CODS, a path with four vertices being the smallest example. We show that the problem of deciding if a graph has a CODS is NP-complete, whereas one can decide in polynomial time if a series-parallel graph contains a CODS. We also examine CODS in various classes of graphs such as grids, complete partite graphs and complements of powers of cycles. In particular, we prove that only 15 grid graphs have CODS.

## 2 Computational aspects

### 2.1 NP-Completeness

**Theorem 1** *It is NP-complete to decide if a graph has a CODS.*

*Proof:* The problem is obviously in NP. To show it is NP-hard, we do a reduction from the NP-complete 1-in-3 3SAT with no negated literals (1-in-3). A figure detailing the most intricate subgraph in the reduction is shown in Figure 1. Let  $F$  be an instance of 1-in-3 with clause set  $C = c_1, c_2, \dots, c_p$  and variable set  $U = u_1, u_2, \dots, u_q$ . Denote the three variables in clause  $c_i$  as  $u_1^i, u_2^i, u_3^i$ .

Construct a graph as follows. For each clause  $c_i$  create a *clause* vertex  $c_i$  and three *variable* vertices  $u_1^i, u_2^i, u_3^i$  that are adjacent to  $c_i$ . Create three *parity* vertices for this clause,  $x_1^i, x_2^i, x_3^i$  where  $x_1^i$  is adjacent to  $u_1^i$  and  $u_2^i$ ;  $x_2^i$  is adjacent to  $u_1^i$  and  $u_3^i$ ; and  $x_3^i$  is adjacent to  $u_2^i$  and  $u_3^i$ . To each parity vertex  $x_j^i$  attach a *parity check* vertex  $p_j^i$  and to parity check vertex  $p_j^i$  attach a vertex  $q_j^i$  and make each  $q_j^i$  adjacent to  $c_i$ . For each  $x_j^i$  create three new vertices  $t_1, t_2, t_3$  (superscripts omitted for clarity) so that  $x_j^i$  is adjacent to  $t_1$ ,  $t_1$  is adjacent to  $t_2$ , and  $t_2$  is adjacent to  $t_3$ . (Note that only one of the three  $t$  type paths is shown in Figure 1.) Also attach to  $c_i$  a path  $w_1^i, w_2^i, w_3^i$  so that  $w_1^i$  is adjacent to  $c_i$ ,  $w_2^i$  is adjacent to  $w_1^i$  and  $w_3^i$  is adjacent to  $w_2^i$ .

Next create three vertices  $f_1^i, f_2^i, f_3^i$  so that  $f_1^i$  is adjacent to  $f_2^i$  and  $f_2^i$  is adjacent to  $f_3^i$ . Connect  $f_1^i$  to both  $u_3^i$  and  $p_3^i$ . Then create three vertices  $g_1^i, g_2^i, g_3^i$  so that  $g_1^i$  is adjacent to  $g_2^i$  and  $g_2^i$  is adjacent to  $g_3^i$ . Connect  $g_1^i$  to both  $u_2^i$  and  $p_1^i$ . Then create three vertices  $h_1^i, h_2^i, h_3^i$  so that  $h_1^i$  is adjacent to  $h_2^i$  and  $h_2^i$  is adjacent to  $h_3^i$ . Connect  $h_1^i$  to both  $u_1^i$  and  $p_2^i$ . Now create a vertex  $e^i$  that is adjacent to  $f_2^i, g_2^i$  and  $h_2^i$ . (Note that the  $f$  and  $h$  type vertices are not shown in Figure 1.)

If variable  $u_j$  appears in clauses  $c_i$  and  $c_k$  create a *consistency* vertex  $y_j^{ik}$  that is adjacent to both  $u_j^i$  and  $u_j^k$ . This is done for every pair of distinct clauses that a given variable appears in. To each  $y_j^{ik}$  attach a two vertex subgraph  $z_1, z_2$  (omitting the subscripts, but being clear that each  $y_j^{ik}$  is attached to distinct such subgraph) where  $y_j^{ik}$  is adjacent to  $z_1$  and  $z_1$  is adjacent to  $z_2$ .

Connect all  $3p$  variable vertices in a complete subgraph. Create a *super-vertex*  $a_1$  and another vertex  $a_2$  where  $a_1$  is adjacent to  $a_2$ . Then connect  $a_1$  to the following vertices:  $w_2^i$ , for all  $i$ ;  $p_j^i$  for all  $i, j$ ;  $q_i^j$ , for all  $i, j$ ;  $f_2^i$ , for all  $i$ ;  $g_2^i$  for all  $i$ ;  $h_2^i$  for all  $i$ ;  $f_3^i$ , for all  $i$ ;  $g_3^i$  for all  $i$ ;  $h_3^i$  for all  $i$ ; and to each  $z_1$  and  $t_2$  vertex. Repeat the same process with another super-vertex  $b_1$ , which has a pendant vertex  $b_2$  attached to it and make  $a_1$  adjacent to  $b_1$ . If  $p$  is even then connect  $a_1$  to  $u_j^i$ , for all  $i, j$ ; else connect both  $a_1$  and  $b_1$  to  $u_j^i$ , for all  $i, j$ . Let the graph constructed so far be called  $G$ .

To complete the construction, if  $a_1$  has odd number of neighbors other than  $a_2$  and  $b_1$  has an odd number of neighbors other than  $b_2$ , attach new vertex  $d$  to  $a_1$  and  $b_1$  with a pendant vertex  $d_1$  connected to  $d$ . Else if  $a_1$  has an odd number of neighbors other than  $a_2$ , then create a copy of  $G$  called  $G'$  and connect  $a_1$  with the corresponding vertex  $a_1'$  in  $G'$ . Likewise, if  $b_1$ , rather than  $a_1$ , has an odd number of neighbors (in  $G$ ) other than  $b_2$ . Let  $G^*$  denote the final graph constructed. It is easy to see that  $G^*$  can be constructed from  $F$  in polynomial time.

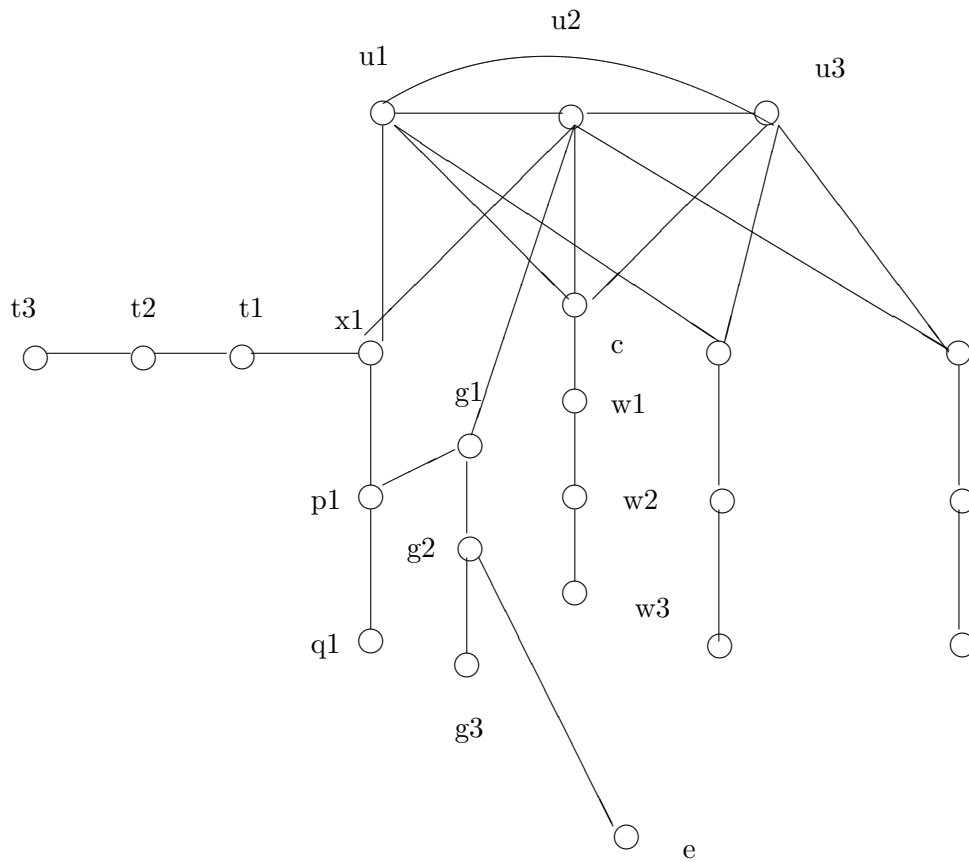


Figure 1: Gadget

We claim that  $F$  is 1-in-3 satisfiable if and only if  $G^*$  has a connected odd dominating set. In the following, a vertex is defined to be adjacent to itself. Suppose  $F$  is 1-in-3 satisfiable, i.e.,  $F$  can be satisfied so that each clause contains exactly one “true” variable. Say  $\{u_1, u_2, \dots, u_r\}$  is the set of variables that are assigned “true” in this satisfying assignment. Then a connected odd dominating set  $D$  is formed by the set of variable vertices  $u_j^i$  where  $u_j^i$  is the unique variable vertex adjacent to clause vertex  $c_i$  corresponding to the “true” variable in clause  $c_i$ , plus the appropriate two  $q_k^i$  vertices for each  $i$  (e.g.,  $q_1^i, q_2^i$  if  $u_1^i \in D$ ), plus the appropriate  $p_j^i$  vertex for each  $i$  (e.g.,  $p_3^i$  if  $u_1^i \in D$ ), plus the appropriate one of  $\{f_2^i, f_3^i\}$  for each  $i$ , plus the appropriate one of  $\{g_2^i, g_3^i\}$  for each  $i$ , plus the appropriate one of  $\{h_2^i, h_3^i\}$  for each  $i$ , plus each of the  $z_1, w_2$ , and  $t_2$  vertices, plus  $a_1$  and  $b_1$ , plus  $d$  if it exists. This set of vertices is clearly connected since  $a_1$  and  $b_1$  are adjacent to each other (and to  $d$ ) and since  $a_1$  is adjacent to each other vertex in  $D$  (except in the case where we create a copy of  $G, G'$ , in which case  $a_1$  is also adjacent to  $a_1'$ ). Note that since  $D$  contains one vertex from each  $u_j^i$  triple of vertices, then exactly one  $p_j^i$  vertex will be in  $D$  for each  $i$ , two  $q_j^i$  vertices will be in  $D$  for each  $i$ , one of  $\{f_2^i, g_2^i, h_2^i\}$  will be in  $D$  for each  $i$ , and two of  $\{f_3^i, g_3^i, h_3^i\}$  will be in  $D$  for each  $i$ .

It is not difficult to see that  $D$  is an odd dominating set: each  $u_j^i$  vertex is adjacent to the other  $u_j^i$  vertices and  $a_1$  and possibly  $b_1$  (depending on the parity of  $p$ ); each  $c_i$  vertex is adjacent to three vertices in  $D$ ; each  $w_1, w_3, t_1, t_3$  vertex is adjacent to one vertex in  $D$ , each  $w_2$  and  $t_2$  vertex is adjacent to three vertices in  $D$ ; each  $y_j^{ik}$  vertex is adjacent to either one or three vertices in  $D$ ; each  $x_j^i$  is adjacent to one vertex in  $D$ ; each  $p_j^i, q_j^i$  is adjacent to three vertices in  $D$ ; each  $z_1$  is adjacent to three vertices in  $D$ ; each  $z_2$  is adjacent to one vertex in  $D$ ; each  $f_1^i, g_1^i$  and  $h_1^i$  vertex is adjacent to one vertex in  $D$ ; each  $f_2^i, g_2^i$  and  $h_2^i$  vertex is adjacent to three vertices in  $D$ ; each  $f_3^i, g_3^i$  and  $h_3^i$  vertex is adjacent to three vertices in  $D$ ; each  $e^i$  is adjacent to one vertex in  $D$ ; each of  $a_2, b_2$  are adjacent to one vertex in  $D$ ; and we constructed  $G^*$  so that  $a_1, b_1$  (and  $d$  if it exists) will have an odd number of neighbors in  $D$ , since every neighbor of  $a_1$  and of  $b_1$  (except for  $a_2$  and  $b_2$ ) is in  $D$ .

Now suppose  $D$  is a connected odd dominating set of  $G^*$ . A few simple observations show that the  $u_j^i$  vertices in  $D$  correspond to a 1-in-3 satisfying assignment for  $F$ . Note that any pendant vertex  $v$  in  $G^*$  cannot be in  $D$ , in fact,  $v$ 's neighbor must be in  $D$ . So no  $x_j^i$  vertex can be in  $D$  because each  $t_2$  vertex must be in  $D$ . Likewise, each  $w_2$  vertex must be in  $D$  and thus no  $w_1$  vertex, nor any  $c_i$  vertex, can be in  $D$ . It follows that at least one of

$u_1^i, u_2^i, u_3^i$  must be in  $D$  for each clause  $c_i$  in order to dominate the vertex  $c_i$ . This is because if none of  $u_1^i, u_2^i, u_3^i$  were in  $D$ , then neither would any of  $q_1^i, q_2^i, q_3^i$  be in  $D$  (since if no  $u_j^i$  vertices were in  $D$ , each of the three  $p_j^i$  would have to be in  $D$  in order to dominate the  $x_j^i$  vertices). There cannot be exactly two vertices from  $u_1^i, u_2^i, u_3^i$  in  $D$ , else in order to avoid an  $x_j^i$  having an even number of neighbors in  $D$ , we would have to add to  $D$  the  $p_j^i$  vertex that is adjacent to  $x_j^i$ . But this implies that  $c_i$  has four neighbors in  $D$  (two  $u_j^i$  type vertices and two type  $q_j^i$  vertices). Note that we cannot have both  $p_j^i$  and  $q_j^i$  in  $D$  as this would mean that each of these are adjacent to four vertices in  $D$  since  $a_1$  and  $b_1$  must also be in  $D$ . Nor can there be three vertices from  $u_1^i, u_2^i, u_3^i$  in  $D$ , because this would force each of three  $p_j^i$  vertices to be in  $D$ . This in turn would force that each of  $f_2^i, g_2^i$  and  $h_2^i$  vertices be in  $D$  (since, for instance, at least one of  $f_2^i, f_3^i$  must be in  $D$  as each  $f_2^i$  and  $f_3^i$  are adjacent to both  $a_1$  and  $b_1$ ). But then  $e^i$  would have no neighbors in  $D$  (and  $e^i$  cannot be in  $D$  else  $D$  would not be connected).

Finally, it is easily seen that the  $y_j^{i,j}$  consistency vertices force that if a variable appears in more than one clause, the corresponding variable vertices in each clause are either both in  $D$  or both not in  $D$ . Thus the  $u_j^i$  vertices in  $D$  correspond to a 1-in-3 satisfying assignment for  $F$ . ■

## 2.2 Series-Parallel graphs

**Proposition 2** *Let  $G$  be a series-parallel graph with  $n$  vertices. There is an  $O(n)$  time algorithm to decide if  $G$  contains a CODS.*

*Proof:* The algorithm is similar to that given in [2] to compute the smallest odd dominating set in a graph. As in that algorithm, we begin by computing a binary tree  $T$  that describes the series and parallel constructions used to build  $G$ . That is, each non-leaf node of  $T$  represents either a series or parallel construction of the two series-parallel graphs that are its children as well as which vertices are the terminals of the resulting graph. Such a tree is called the *parse tree* of the series-parallel graph [2]. The remainder of the algorithm is a simple dynamic programming algorithm, whose correctness follows by induction (the details of which are straightforward and omitted). Working from the leaf level of  $T$  upwards to the root level, we process a node in  $T$  as follows. We store at each node  $v$  of  $T$  additional information as we process  $T$ , namely, whether there exists a CODS of the series-parallel graph  $G_v$  that is described by the subtree of  $T$  whose root is  $v$  and also which of the terminals of  $G_v$  can possibly be in such a CODS. Let  $y$  and  $z$  be the

children of  $v$  in  $T$ . Let  $y_1, y_2$  be the two terminals of  $G_y$  and  $z_1, z_2$  be the two terminals of  $G_z$ .

If  $v$  represents a series construction of  $G_y$  with  $G_z$  (with  $y_2$  identified with  $z_1$ ), then there is a CODS of  $G_v$  if and only if there is a CODS of  $G_y$  containing  $y_2$  and a CODS of  $G_z$  containing  $z_1$ . We must record at  $v$  whether or not there is a CODS of  $G_v$  containing  $y_1$  (which is the case if there is a CODS of  $G_v$  and there is a CODS of  $G_y$  containing  $y_1$ ) and if there is a CODS of  $G_v$  containing  $z_2$  (which is the case if there is a CODS of  $G_v$  and there is a CODS of  $G_z$  containing  $z_2$ ).

If  $v$  represents a parallel construction of  $G_y$  and  $G_z$ , then  $G_v$  has a CODS if and only if  $G_y$  has a CODS containing  $y_1$  and  $y_2$  and  $G_z$  has a CODS containing  $z_1$  and  $z_2$ , in which case  $G_v$  has a CODS containing both its terminals. Note that we do not need to consider the case when both  $G_y$  and  $G_z$  are  $P_2$ 's, since  $G$  is required to be a simple graph. ■

### 2.3 $k$ -exclusive graphs

$k$ -exclusive graphs were defined in [6].

**Definition** A graph is  $k$ -exclusive if its vertices can be ordered  $v_1, \dots, v_n$  such that for every  $j > k$ ,  $v_j$  is the unique neighbor in  $\{v_j, \dots, v_n\}$  of at least one vertex in  $\{v_1, \dots, v_{j-1}\}$ .

Note that this class contains several well-known classes of graphs including the  $k^{\text{th}}$  power of paths and cycles and grids of dimension  $k \times m$ . The following was proved in [6].

**Proposition 3** *Let  $G$  be a  $k$ -exclusive graph with vertex ordering  $v_1, \dots, v_n$  realizing the  $k$ -exclusiveness. Then for every  $j \geq 1$ , the following hold:*

1. *At most  $k$  vertices in  $\{v_1, \dots, v_j\}$  have neighbors in  $\{v_{j+1}, \dots, v_n\}$ .*
2. *Vertex  $v_j$  is adjacent to at most  $k$  vertices preceding it.*

We now describe the CODS algorithm for  $k$ -exclusive graphs.

**Proposition 4** *Let  $G$  be a  $k$ -exclusive graph with  $n$  vertices and its  $k$ -exclusive vertex ordering given. Then a CODS can be found or be determined not to exist in time  $O(2^k k^3 n)$ .*

*Proof:* Let  $v_1, \dots, v_n$  be a vertex order realizing the  $k$ -exclusiveness of  $G$ . Using an “exhaustive-search” strategy, we shall construct a candidate dominating set,  $D$ , by considering all possible  $2^k$  combinations of vertices

$\{v_1, \dots, v_k\}$ . Let  $f$  denote the characteristic function of  $D$ . For each combination of these  $k$  vertices, we verify that each vertex  $v$  in  $\{v_1, \dots, v_k\}$  that has no neighbor in  $\{v_{k+1}, \dots, v_n\}$  satisfies  $|N[v] \cap D| \equiv b \pmod{m}$ . If this congruence is satisfied for all such vertices in  $\{v_1, \dots, v_k\}$ , we can continue. Otherwise, the initial combination is illegal and the next combination is tested. If the initial combination is legal, we consider (in increasing order of index) vertex  $v_j$ , which is the unique neighbor in  $\{v_j, \dots, v_n\}$  of at least one vertex in  $\{v_1, \dots, v_{j-1}\}$  and, by Proposition 3, of at most  $k$  vertices, in  $\{v_1, \dots, v_{j-1}\}$ . The possible value, 0 or 1, of  $f(v_j)$  is completely determined by those vertices in  $\{v_1, \dots, v_{j-1}\}$  for which  $v_j$  is the unique neighbor in the set  $\{v_j, \dots, v_n\}$ . Add  $v_j$  to  $D$  (or not) if its addition (omission) properly satisfies the parity domination constraints for these vertices and the requirement that  $D$  be connected. If so, proceed to  $v_{j+1}$ , otherwise consider the next initial combination. If this algorithm does not terminate with a successful construction of  $D$ , then we infer no CODS exists.

Testing each initial combination takes  $O(k^2)$  time. As we can maintain during the algorithm the value  $p(v) \equiv |N[v] \cap D| \pmod{2}$  for every vertex  $v$  already visited, we can decide “legality” of the value of  $f(v_j)$  in  $O(k)$  time: updating the  $p(v)$  values that may be changed can be done in  $O(k)$  time as  $v_j$  has at most  $k$  neighbors. Thus the running time of the algorithm is as claimed. ■

### 3 Combinatorial aspects

#### 3.1 Grid graphs

Much of the attention on even and odd dominating sets has focused on grid graphs [2, 6, 9, 10, 11]. We show in this section that only a handful of grids have CODS.

Denote by  $G_{m,n}$  the grid with  $m$  rows and  $n$  columns, where  $n \geq m$ . The vertices of the grid are the pairs  $(i, j)$ ,  $i = 1, \dots, m$  and  $j = 1, \dots, n$ .

**Theorem 5** *For all  $7 < m \leq n$ , the grid  $G_{m,n}$  does not have a CODS. There are precisely 15 finite grids that have CODS.*

*Proof:* We first prove that for all  $21 \leq m \leq n$ , the grid  $G_{m,n}$  does not have a CODS. Fix a grid  $G_{m,n}$  with  $m \geq 21$ . Let  $A = \{(1, 1), (2, 1), \dots, (21, 1)\}$  denote the first 21 vertices in the first column of  $G_{m,n}$ . We will show that for any possible choice of a subset  $B \subset A$ , and any choice of  $D \subset V(G_{m,n})$

satisfying  $D \cap A = B$ , then  $D$  is not a CODS. Consequently,  $G_{m,n}$  has no CODS. Notice that there are precisely  $2^{21}$  choices for  $B$ .

Fix a choice of  $B$ . We claim that if  $D$  is an odd dominating set of  $G_{m,n}$  which satisfies  $D \cap A = B$  then the membership in  $D$  of all pairs  $(i, j)$  for  $j = 2, \dots, 21$  and  $i = 1, \dots, 22 - j$  is determined. Indeed, for  $j = 2$  we have that for all  $i = 1, \dots, 20$ , the vertex  $(i, 2)$  is in  $D$  if and only if  $|D \cap \{(i, 1), (i-1, 1), (i+1, 1)\}|$  is even. Similarly, for  $j = 3$  we have that for all  $i = 1, \dots, 19$ , the vertex  $(i, 3)$  is in  $D$  if and only if  $|D \cap \{(i, 2), (i-1, 2), (i+1, 2), (i, 1)\}|$  is even, and so on until  $j = 21$ . We have shown that the membership in  $D$  of all the 231 vertices in the “lower left” triangle with side length 21 is determined.

Let  $C \subset D$  denote the vertices of this “lower left” triangle that belong to  $D$ . Namely,  $C = \{(i, j) \in D : 1 \leq i \leq 21, 1 \leq j \leq 22 - i\}$ . Notice that  $B \subset C$  and notice that  $B$  determines  $C$  uniquely. We claim that the subgraph of  $G_{m,n}$  induced by  $C$  has (at least one) connected component  $W$  consisting of vertices not belonging to the external diagonal of  $C$ . Namely,  $W$  consists only of vertices from  $\{(i, j) : 1 \leq i \leq 20, 1 \leq j \leq 21 - i\}$ . This implies that  $W$  is also a connected component of the subgraph of  $G_{m,n}$  induced by  $D$ . In particular,  $D$  does not induce a connected subgraph of  $G_{m,n}$ , completing the proof.

The computer program in the Appendix verifies our last claim. The program generates all  $2^{21}$  possible  $B$  (the function “nextB” in the program). For any  $B$  generated by our program, the program computes  $C$  (the function “buildC”). Then, the program checks whether any vertex of  $C$  is reachable from the external diagonal (the function “checkC”). Any vertex of  $C$  not reachable from the external diagonal demonstrates the existence of  $W$ , and hence such a  $B$  is discarded. If all vertices of  $C$  are reachable from the boundary, then  $C$  is output (the function “printC”). It turns out that the choice of the constant 21 is the lowest number that causes the program to output nothing. Figure 2 demonstrates a plausible  $B$  and  $C$  that have no  $W$  in case we try to replace 21 by 20. Notice that all vertices of  $C$  are reachable from the external diagonal. In fact, this example is *unique* up to transposing the columns and rows, (namely, the output of the program in the case of the constant 20 consists of two plausible  $C$ : the one from Figure 2 and its transpose).

Finally, we need to show how to reduce the constant 21 to the constant 8, as stated in the theorem. For each *fixed*  $m = 1, \dots, 20$  we perform the following algorithm. Consider a grid  $G_{m,\infty}$ . Notice the obvious fact that any odd dominating set  $D$  of  $G_{m,\infty}$  is determined by the vertices of  $D$  belonging to the first column. Thus, there are precisely  $2^m$  possible choices for  $D$ . Fix

a subset  $B$  of the vertices of the first column that corresponds to the vertices of  $D$  belonging to the first column. We now sequentially construct, using our program, the vertices of  $D$  belonging to columns 2, 3, 4, ... We do this until we reach a column consisting of no vertices of  $D$  (i.e., a column of zeroes, which we call a *null* column). Indeed, an easy periodicity argument shows that we must always eventually reach a null column (our program shows that the index of the null column never exceeds 2000 in case  $m \leq 20$ ). Once the null column is reached, there is no point to continue since we are searching for connected  $D$ . Thus, when reaching the null column we check whether the resulting  $D$  is connected. If so, we output  $D$ , but only in the case where the index of the null column is greater than  $m$  (to avoid multiplicities we assume  $m$  is the smaller dimension). In case the index of the null column is equal to  $m$ , we output  $D$  only if the column before the null column is completely within  $D$  (since, in this case, the null column is allowed to be part of the grid). Our program functions “nextSmallB”, “nextColumn”, “checkSmallC” and “PrintSmallC” perform the operations mentioned here. It turns out that the program outputs nothing for  $m = 8, \dots, 20$ .

Figure 2

```

11101100000111001110
1011110111010100101
100010110111010010
10011010000011001
1001011110011000
100101101001011
10011000100101
1000110010011
101101001000
11110100101
0010110011
011010000
01011111
0101100
011001
00111
1100
111
01
1

```

Figure 3

```

11011011
11111111
01011010
11000011
10000001
11111111
00000000

```

For  $m = 1, \dots, 7$  the program outputs all grids that have CODS, and their respective CODS (in some cases there are more than one CODS). In fact, the following grids have CODS:  $G_{1,1}, G_{1,2}, G_{1,3}, G_{2,2}, G_{2,3}, G_{2,4}, G_{3,4}, G_{3,5}, G_{3,6}, G_{4,4}, G_{4,5}, G_{6,7}, G_{6,8}, G_{6,9}, G_{7,8}$ . Figure 3 shows one of the two possible CODS of  $G_{7,8}$ . ■

Notice that an immediate corollary of the theorem is that if either  $m$  or  $n$  (or both) is infinite, then the resulting infinite grid has no CODS.

A detailed analysis of length of the periodicity of the related recurrence for even dominating sets of grids can be found in [9, 11].

### 3.2 Complements of powers of cycles

Odd dominating sets in powers of cycles and their complements were studied in [6]. Denote these as  $C_n^k$  and  $\overline{C_n^k}$ , respectively, where  $n \geq 3, k \geq 1$ . It is easy to see that  $C_n^k$  is either complete or an Eulerian graph and thus always has a CODS, likewise  $\overline{C_n^k}$  is either isolated or Eulerian when  $n$  is odd (notice that  $C_n^k$  is connected only for  $n \geq 2k + 3$ ). The situation is not so clear when  $n$  is even; the following represent a partial characterization.

**Theorem 6** *Let  $n, k$  be positive integers such that  $n \geq 2k + 3$ . The graph  $\overline{C_n^k}$  has a CODS in the following cases:*

- 1)  $\overline{C_n^k}$  has a CODS if  $n \equiv 1 \pmod{2}$ .
- 2)  $\overline{C_n^k}$  has a CODS if  $n \equiv 2 \pmod{4}$ .
- 3)  $\overline{C_n^1}$  has a CODS and  $\overline{C_n^2}$  has a CODS.
- 4)  $\overline{C_n^k}$  and  $\overline{C_n^{k+1}}$  have CODS if  $k$  is odd and  $n \equiv 0 \pmod{2(k+1)}$ .
- 5)  $\overline{C_{12k}^{4k}}$  has a CODS.

*Proof:*

1. If  $n \equiv 1 \pmod{2}$  then the graph is Eulerian and we simply take  $V(G)$  as a CODS.
2. If  $n \equiv 2 \pmod{4}$  then label the vertices  $0, 1, 2, \dots, n-1$  and take for the CODS the set of vertices with even label. Notice that  $|D| = n/2$  is odd and, by symmetry, every vertex is not a neighbor of an even number of vertices in  $D$ . Hence,  $D$  is an odd dominating set. Furthermore,  $D$  induces a connected subgraph as  $D$  has the following Hamiltonian cycle:  $0, n/2 + 1, 2, n/2 + 3, \dots, n/2 - 3, n - 2, n/2 - 1, 0$ .
3. We must show that  $\overline{C_n^1}$  has a CODS and  $\overline{C_n^2}$  has a CODS. By the previous cases, we only need to check the case  $n \equiv 0 \pmod{4}$ . Label

the vertices  $0, 1, 2, \dots, n-1$  and take for the CODS the set of vertices  $D = \{z : z \equiv 0, 1 \pmod{4}\}$ . It is straightforward to verify that  $D$  is a CODS.

4. We must show that  $\overline{C_n^k}$  and  $\overline{C_n^{k+1}}$  have CODS if  $k$  is odd and  $n \equiv 0 \pmod{2(k+1)}$ . Indeed, label the vertices  $0, 1, 2, \dots, n-1$  and consider this labeling over  $Z_{2(k+1)}$ . Take the CODS to be the set of vertices  $D = \{z : z \equiv 0, 1, 2, \dots, k \pmod{2(k+1)}\}$ . It is not hard to verify that  $D$  is a CODS.
5. We must show that  $\overline{C_{12k}^{4k}}$  has a CODS. Label the vertices  $0, 1, \dots, 12k-1$  and consider the set of vertices  $D = V - \{0, 4k, 8k\}$ . So  $|D| = 12k-3$  and the verification that  $D$  is indeed a CODS is easy. ■

Let  $01^n$  denote the sequence consisting of one “0” followed by  $n$  “1”’s and in general let  $a^n$  denote the sequence  $a$  repeated  $n$  times.

**Theorem 7** *Let  $n$  and  $k$  be positive integers, where  $n > 8k$ . Suppose  $k$  is odd and  $n$  is even. Let  $t = \gcd(n - 2(k+1), k+1, n)$ . If  $(n - 2(k+1))/t$  is odd then the sequence  $(01^{t-1})^{n/t}$  is the characteristic function of a CODS of  $\overline{C_n^k}$  and  $\overline{C_n^{k+1}}$ .*

We provide two examples before giving the proof (in the first example  $n \leq 8k$ , but the idea is the same).

Example 1:  $n = 12, k = 3$ . Then  $t = \gcd(12 - 8, 12, 4) = 4$  and  $(12 - 8)/4 = 4/4 = 1$ . Thus,  $011101110111$  is a CODS for  $\overline{C_{12}^3}$  and  $\overline{C_{12}^4}$ .

Example 2:  $n = 36, k = 3$ . Then  $t = \gcd(n - 2(k+1), k+1, n) = \gcd(28, 4, 36) = 4$  and  $(n - 2(k+1))/t = 28/4 = 7$  is odd. Hence  $(0111)^9$  is a CODS for  $\overline{C_{36}^3}$  and  $\overline{C_{36}^4}$ .

*Proof of Theorem 7:* Observe that  $n - 2(k+1)$  is just one less than the degree of  $\overline{C_n^k}$ . We also observe also that the length of the sequence  $01^{t-1}$  is even (as  $n$  and  $k+1$  are even) and thus  $t-1$  is odd and so the number of “1”’s in  $(n - 2(k+1))/t$  repetitions of the sequence is  $(n - 2(k+1))(t-1)/t$ , which is odd.

Take  $D = V - \{z : z \equiv 0 \pmod{t}\}$ . We claim  $D$  is a CODS of  $\overline{C_n^k}$ . We first show the odd-domination property. The vertex  $v_0$  is adjacent to precisely  $(n - 2(k+1))/t$  repetitions of  $01^{t-1}$  plus an extra 0 as its degree is  $n - 2(k+1) + 1$  and as  $t|(k+1)$ . So the first edge emanating from  $v_0$ , going clockwise around the cycle, is to a vertex  $v_j$ , where  $j \equiv 0 \pmod{t}$  and hence  $v_j$  is not in  $D$ . For the same reason, the last edge out of  $v_0$  is to a

vertex  $v_r$ , where  $r \equiv 0 \pmod t$ , and thus  $v_r$  is not in  $D$ . Hence  $v_0$  has an odd number of “1”’s in its closed neighborhood,  $(n - 2(k + 1))(t - 1)/t$  to be exact. The vertices  $v_1, \dots, v_{t-1}$  are not adjacent to the first 0 that is adjacent to  $v_0$  (which is  $v_j$ ), but each are adjacent to two additional “1”’s, as compared to  $v_0$  (one of these additional “1”’s is themselves).

As to  $\overline{C_n^{k+1}}$ , it follows that, in comparison with  $v_0$  in  $\overline{C_n^k}$ ,  $v_0$  in  $\overline{C_n^{k+1}}$  loses the first and the last “0” in its neighborhood (denoted  $v_j$  and  $v_r$  above) and so, as before,  $v_0$ ’s closed neighborhood contains an odd number of members of  $D$ , while each of  $v_1, \dots, v_{t-1}$  add one “0” ( $v_r$ ) and lose one “1” (relative to  $v_0$ ), but they themselves add “1” to their closed neighborhood and again have an odd number of (closed) neighbors in  $D$ .

Finally,  $D$  is connected since  $n > 8k$  and the fact that  $t \geq 2$  implies that the subgraph induced by  $D$  has minimum degree at least half its order. ■

It is plausible that Theorem 7 holds for all  $n \geq 2k + 5$ . One might guess that  $n \geq 2k + 3$  is a necessary and sufficient condition for  $\overline{C_n^k}$  to have a CODS. This however is not the case, since  $\overline{C_{16}^5}$ ,  $\overline{C_{16}^6}$  and  $\overline{C_{24}^{10}}$  have no CODS, as we verified using a computer.

### 3.3 Complete partite graphs

**Proposition 8** *A complete  $q$ -partite graphs  $K_{a_1, \dots, a_q}$ ,  $q \geq 2$  has a CODS if and only if*

- (a)  $(a_1, \dots, a_q) \equiv (0, 0, \dots, 0) \pmod 2$ ; or
- (b) there exist distinct  $i, j, k$  such that  $(a_i, a_j, a_k) = (1, 1, 1) \pmod 2$ ; or
- (c)  $a_i = 1$  for some  $1 \leq i \leq q$ .

*Proof:* Clearly the graphs in (a), (b) and (c) have CODS. Now assuming our graph is not one of (a), (b) or (c), then it must have one or two vertex classes with odd cardinality at least 3. Since vertices in the same class are transitive, each class is either completely in or completely out of an odd dominating set. In case there is precisely one class with odd cardinality at least 3, it is easily checked that whether this class is in or out of a dominating set, such a dominating set is not a CODS. Similarly, if there are two classes with odd cardinality at least 3, it is easily checked that whether none, one of, or both of the classes are in a dominating set, such a dominating set is not a CODS. ■

### 3.4 Powers of paths

Let  $P_n^k$  denote the  $k^{\text{th}}$  power of the path on  $n$  vertices.

**Proposition 9** *If  $2k+1 \geq n$ , then  $P_n^k$  has a CODS (a vertex of the centroid of the path). If  $2k+1 < n$  then  $P_n^k$  has no CODS.*

*Proof:* If  $2k+1 \geq n$ , the vertex in position  $\lfloor n/2 \rfloor$  is connected to all other vertices and hence constitutes a CODS. So we assume  $2k+1 < n$ . Let  $v_1, \dots, v_n$  be the vertices of the path, and suppose  $D$  is a CODS of  $P_n^k$ . Define  $q(v) = 1$  if  $v \in D$  and  $q(v) = 0$  otherwise. Clearly  $q(v_1) + q(v_2) + \dots + q(v_{k+1}) = |N[v_1] \cap D| \equiv 1 \pmod{2}$ . Also  $q(v_1) + q(v_2) + \dots + q(v_{k+1}) + q(v_{k+2}) = |N[v_2] \cap D| \equiv 1 \pmod{2}$ , forcing  $q(v_{k+2}) = 0$ . And  $q(v_1) + q(v_2) + \dots + q(v_{k+2}) + q(v_{k+3}) = |N[v_3] \cap D| \equiv 1 \pmod{2}$ , forcing  $q(v_{k+3}) = 0$ . This pattern continues until  $q(v_1) + q(v_2) + \dots + q(v_{k+2}) + q(v_{2k+1}) = |N[v_{k+1}] \cap D| \equiv 1 \pmod{2}$ , forcing  $q(v_{2k+1}) = 0$ . But then the  $k$  consecutive vertices  $v_{k+2}, v_{k+3}, \dots, v_{2k+1}$  are not in  $D$  and hence  $D$  is not connected since  $n > 2k+1$ . ■

### 3.5 Graph products and cubes

**Fact 10** *if  $G$  has a CODS and  $H$  is Eulerian then  $G \times H$  has a CODS.*

For example, we can use this fact to deduce that for all  $d \geq 1$ , the  $d$ -dimensional cube  $Q_d$  has a CODS. Indeed, it is trivial for  $d = 1$ . For  $d$  even,  $Q_d$  is Eulerian. For  $d$  odd, we use the fact that  $Q_{d+1} = K_2 \times Q_d$  and Fact 10.

## 4 Concluding remarks and open problems

Characterizing which complements of powers of cycles has CODS is a problem that remains. We state three problems:

1. The general problem: For which  $n$  and  $k$ , such that  $n \geq 2k+3$ , does  $\overline{C_n^k}$  have a CODS?
2. Is it true that  $\overline{C_n^3}$  and  $\overline{C_n^4}$  always has a CODS provided  $n \geq 2k+3$ ?
3. Is it true that  $\overline{C_n^k}$  has CODS for  $n \equiv 4 \pmod{8}$  provided  $n \geq 2k+3$ ?

It is of interest to resolve the complexity of the CODS problem for several classes of graphs: interval graphs, bipartite graphs, planar graphs, and partial  $k$ -trees for  $k > 2$ .

## References

- [1] A. Amin, L. Clark, and P. Slater (1998), Parity Dimension for Graphs, *Discrete Mathematics*, vol. 187, pp. 1-17
- [2] A. Amin and P. Slater (1992), Neighborhood Domination with Parity Restriction in Graphs, *Congressus Numerantium*, vol. 91, pp. 19-30
- [3] A. Amin and P. Slater (1996), All Parity Realizable Trees, *J. Comb. Math. and Comb. Comput.*, vol. 20, pp. 53-63
- [4] Y. Caro (1996), Simple Proofs to Three Parity Theorems, *Ars Comb.*, 42, pp. 175-180
- [5] Y. Caro and W. Klostermeyer (2003), The Odd Domination Number of a Graph, *J. Comb. Math. Comb. Comput.*, vol. 44, pp. 65-84
- [6] Y. Caro, W. Klostermeyer, and J. Goldwasser (2001), "Odd and Residue Domination Numbers of a Graph," *Discussiones Mathematicae Graph Theory*, vol. 21, pp. 119-136
- [7] M. Conlon, M. Falidas, M. Forde, J. Kennedy, S. McIlwaine, and J. Stern (1999), Inversion Numbers of Graphs, *Graph Theory Notes of New York*, vol. XXXVII, pp. 43-49
- [8] R. Cowen, S. Hechler, J. Kennedy, and A. Ryba (1999), Inversion and Neighborhood Inversion in Graphs, *Graph Theory Notes of New York*, vol. XXXVII, pp. 38-42
- [9] J. Goldwasser, W. Klostermeyer, and G. Trapp, (1997), Characterizing Switch-Setting Problems, *Linear and Multilinear Algebra*, vol. 43, pp. 121-135
- [10] J. Goldwasser and W. Klostermeyer (1997), Maximization Versions of "Lights Out" Games in Grids and Graphs, *Congressus Numerantium*, vol. 126, pp. 99-111
- [11] J. Goldwasser, W. Klostermeyer, G. and H. Ware (2002), Fibonacci Polynomials and Parity Domination in Grid Graphs, *Graphs and Combinatorics*, vol. 18, pp. 271-283
- [12] M. Halldorsson, J. Kratochvil, J. Telle (1999), "Mod-2 Independence and Domination in Graphs," *Proceedings Workshop on Graph-Theoretic Concepts in Computer Science '99*, Ascona, Switzerland, Springer-Verlag Lecture Notes in Computer Science

- [13] T. Haynes, S. Hedetniemi, and P. Slater (1998), **Fundamentals of Domination in Graphs**, Marcel Dekker, New York
- [14] K. Sutner (1989), Linear Cellular Automata and the Garden-of-Eden, *The Mathematical Intelligencer*, vol. 11, no. 2, pp. 49-53

## Appendix

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

const numVertices = 21;
const maxColumns = 2000;
int grid[numVertices][numVertices];
int smallGrid[numVertices][maxColumns];

bool nextB()
{
    int i;
    for (i=0; i < numVertices; i++)
    {
        grid[i][0] = 1-grid[i][0];
        if (grid[i][0] == 1)
            return true;
    }
    return false;
}

void buildC()
{
    int i,j,sum;
    for(j=1;j< numVertices; j++)
    {
        for(i=0;i < numVertices-j;i++)
        {
            sum = grid[i][j-1]+grid[i+1][j-1];
            if (i > 0)
                sum += grid[i-1][j-1];
            if (j > 1)
                sum += grid[i][j-2];
            if (sum % 2 == 0)
                grid[i][j] = 1;
            else
                grid[i][j] = 0;
        }
    }
}
```

```

        }
    }
}

void printC()
{
    for (int i=0; i < numVertices; i++)
    {
        for (int j=0; j < numVertices-i; j++)
            printf("%d",grid[i][j]);
        printf("\n");
    }
}

bool checkC()
{
    static int bfsQueue[numVertices*numVertices][2];
    static int mark[numVertices+1][numVertices+1];
    int head, tail;
    int i,j;

    for(i=0;i<=numVertices;i++)
        for(j=0;j <= numVertices;j++)
            mark[i][j]=0;

    for(i=0;i<=numVertices;i++)
    {
        bfsQueue[i][0]=i;
        bfsQueue[i][1]=numVertices-i;
        mark[i][numVertices-i]=1;
    }
    head=0;
    tail=numVertices;

    while (head <= tail)
    {
        i=bfsQueue[head][0];
        j=bfsQueue[head][1];
        head++;
        if ((i > 0) && (grid[i-1][j] == 1) &&

```

```

        (mark[i-1][j] == 0)
    {
        mark[i-1][j]=1;
        bfsQueue[tail+1][0]=i-1;
        bfsQueue[tail+1][1]=j;
        tail++;
    }
    if ((j > 0) && (grid[i][j-1] == 1) &&
        (mark[i][j-1] == 0))
    {
        mark[i][j-1]=1;
        bfsQueue[tail+1][0]=i;
        bfsQueue[tail+1][1]=j-1;
        tail++;
    }
    if ((i+j < numVertices) && (grid[i+1][j] == 1) &&
        (mark[i+1][j] == 0))
    {
        mark[i+1][j]=1;
        bfsQueue[tail+1][0]=i+1;
        bfsQueue[tail+1][1]=j;
        tail++;
    }
    if ((i+j < numVertices) && (grid[i][j+1] == 1) &&
        (mark[i][j+1] == 0))
    {
        mark[i][j+1]=1;
        bfsQueue[tail+1][0]=i;
        bfsQueue[tail+1][1]=j+1;
        tail++;
    }
}
for (i=0; i < numVertices;i++)
    for(j=0;j < numVertices-i; j++)
        if ((grid[i][j]== 1) && (mark[i][j] == 0))
            return false;
return true;
}

```

```

bool nextSmallB(int m)

```

```

{
    int i;
    for (i=0; i < m; i++)
    {
        smallGrid[i][0] = 1-smallGrid[i][0];
        if (smallGrid[i][0] == 1)
            return true;
    }
    return false;
}

```

```

bool nextColumn(int m, int c)
{
    int i,sum;
    for (i=0; i < m; i++)
    {
        sum = smallGrid[i][c-1];
        if (i > 0)
            sum += smallGrid[i-1][c-1];
        if (i < m-1)
            sum += smallGrid[i+1][c-1];
        if (c > 1)
            sum += smallGrid[i][c-2];
        if (sum % 2 == 0)
            smallGrid[i][c] = 1;
        else
            smallGrid[i][c] = 0;
    }
    for (i = 0; i < m; i++)
        if (smallGrid[i][c] == 1)
            return true;
    return false;
}

```

```

void printSmallC(int m, int c)
{
    int i,j;

    if (m <= c)
    {

```

```

        for(i=0; i < m; i++)
        {
            for(j=0; j < c ; j++)
                printf("%d",smallGrid[i][j]);
            printf("\n");
        }
        printf("\n");
    }
    for (i=0; i < m; i++)
        if (smallGrid[i][c-1] == 0)
            return;
    if (m <= c+1)
    {
        for(i=0; i < m; i++)
        {
            for(j=0; j < c ; j++)
                printf("%d",smallGrid[i][j]);
            printf("0\n");
        }
        printf("\n");
    }
}

```

```

bool checkSmallC(int m, int c)
{
    static int bfsQueue[numVertices*maxColumns] [2];
    static int mark[numVertices] [maxColumns];
    int head, tail;
    int i,j;

    for(i=0;i< m ;i++)
        for(j=0;j < c;j++)
            mark[i][j]=0;

    for(i=0; i < m; i++)
        if (smallGrid[i][c-1] == 1)
        {
            bfsQueue[0][0]=i;
            bfsQueue[0][1] = c-1;
            mark[i][c-1] = 1;
        }
}

```

```

    break;
}
head=0;
tail=0;

while (head <= tail)
{
    i=bfsQueue[head][0];
    j=bfsQueue[head][1];
    head++;
    if ((i > 0) && (smallGrid[i-1][j] == 1) &&
        (mark[i-1][j] == 0))
    {
        mark[i-1][j]=1;
        bfsQueue[tail+1][0]=i-1;
        bfsQueue[tail+1][1]=j;
        tail++;
    }
    if ((i < m-1) && (smallGrid[i+1][j] == 1) &&
        (mark[i+1][j] == 0))
    {
        mark[i+1][j]=1;
        bfsQueue[tail+1][0]=i+1;
        bfsQueue[tail+1][1]=j;
        tail++;
    }
    if ((j > 0) && (smallGrid[i][j-1] == 1) &&
        (mark[i][j-1] == 0))
    {
        mark[i][j-1]=1;
        bfsQueue[tail+1][0]=i;
        bfsQueue[tail+1][1]=j-1;
        tail++;
    }
    if ((j < c-1) && (smallGrid[i][j+1] == 1) &&
        (mark[i][j+1] == 0))
    {
        mark[i][j+1]=1;
        bfsQueue[tail+1][0]=i;
        bfsQueue[tail+1][1]=j+1;
    }
}

```

```

        tail++;
    }
}
for(i=0;i< m ;i++)
    for(j=0;j < c;j++)
        if ((smallGrid[i][j]== 1) && (mark[i][j] == 0))
            return false;
return true;
}

void checkSmallGrids()
{
    int i,m,c;
    bool result;

    for (m=1; m < numVertices; m++)
    {
        printf("Grids with small side length %d\n",m);
        for(i=0; i < m; i++)
            smallGrid[i][0] = 0;

        do
        {
            c = 1;
            for (;;)
            {
                if (!nextColumn(m,c))
                {
                    if (checkSmallC(m,c))
                        printSmallC(m,c);
                    break;
                }
                c++;
                if (c > 1998)
                {
                    printf("%d\n",c);
                    getch();
                }
            }
        } while (nextSmallB(m));
    }
}

```

```
        getch();
    }

int main(int argc, char* argv[])
{
    int i;

    for(i=0; i < numVertices; i++)
        grid[i][0] = 0;
    i = 0;
    do
    {
        buildC();
        if (checkC())
        {
            i++;
            printC();
        }
    } while (nextB());
    printf("Candidate CODS: %d\n",i);
    getch();
    printf("Now checking small grids\n");
    checkSmallGrids();
    return 0;
}
```