HPC acceleration of large (min, +) matrix products to compute domination-type parameters in graphs

Ester M. Garzón[1,3] · **José Antonio Martínez**[1,3] · **Juan José Moreno**[1,3] ·
María Luz Puertas[2,3]

Abstract

The computation of the domination-type parameters is a challenging problem in Cartesian product graphs. We present an algorithmic method to compute the 2-domination number of the Cartesian product of a path with small order and any cycle, involving the (min, +) matrix product. We establish some theoretical results that provide the algorithms necessary to compute that parameter, and the main challenge to run such algorithms comes from the large size of the matrices used, which makes it necessary to improve the techniques to handle these objects. We analyze the performance of the algorithms on modern multicore CPUs and on GPUs and we show the advantages over the sequential implementation. The use of these platforms allows us to compute the 2-domination number of cylinders such that their paths have at most 12 vertices.

Keywords (min, +) matrix powers · OpenMP · GPU · 2-domination

---

The four authors contributed equally to this work.

✉ María Luz Puertas
  mpuertas@ual.es

  Ester M. Garzón
  gmartin@ual.es

  José Antonio Martínez
  jmartine@ual.es

  Juan José Moreno
  juanjomoreno@ual.es

[1] Department of Computer Sciences, Universidad de Almería, Almería, Spain

[2] Department of Mathematics, Universidad de Almería, Almería, Spain

[3] Agrifood Campus of International Excellence (ceiA3), Universidad de Almería, Almería, Spain

1 Introduction

The $(\min, +)$ matrix algebra [1], also called tropical algebra, replaces addition and multiplication with minimization and addition, respectively. The use of this algebra is currently in expansion and it is involved in several disciplines of great interest, for instance finite automata [1], statistics [2], phylogenetics [3], optimization of graph parameters [4], integer programming [5], and other optimization problems [6]. However, the computational demands of such computations are unapproachable when the dimensions of the corresponding matrices are large. To overcome this drawback the modern multicore CPUs and GPUs can be exploited as High-Performance Computing (HPC) platforms to accelerate and widen the dimensions of such operations. In this work, the analysis of the domination-type parameters in graphs is chosen as an interesting example where sequences of large $(\min, +)$ matrix products are involved.

The use of graphs as a tool to model problems in networks has been widely studied. Among such problems, the efficient location of resources in a network can be approached by means of the domination-type parameters in graphs. A dominating set in a graph G is a vertex subset S such that each vertex not in S has at least one neighbor in it. The domination number of G, denoted by $\gamma(G)$, is the cardinal of a minimum dominating set. We refer to [7] for general information about these topics and, in particular, about their applications to network problems. Among the variations of this concept that can be found in the literature, we focus on the 2-domination. A 2-*dominating set* is vertex subset $S \subseteq V(G)$ such that each vertex not in S has at least two neighbors in it. The 2-*domination number* $\gamma_2(G)$ is the minimum cardinal of a 2-dominating set of G [8]. Some interesting applications of the 2-domination in graphs such as the optimization of fault tolerant sensor networks, the facility location problem and the data collection problem can be found in [9]. Given a graph G and a positive integer $k \leq |V(G)|$, the decision problem "Is there a dominating set of G with at most k vertices?" is NP-complete [10], even in bipartite and chordal graphs. However, it has been shown to be polynomial in trees and interval graphs [7]. In a similar way, the 2-domination decision problem is to decide whether G has a 2-dominating set of cardinal at most $k \leq |V(G)|$. It is known that it is an NP-complete problem [11], again even in bipartite and chordal graphs [12]. Moreover, linear-time algorithms to compute this parameter in trees and series-parallel graphs can also be found in [11].

A family of interest for the domination-type parameters are the Cartesian product graphs since the Vizing's conjecture was formulated [13]. This conjecture proposes a general inequality that relates the domination number of both a Cartesian product graph and its factors. This conjecture is still open and a survey about this subject can be found in [14], while a recent new approach is in [15]. Recall that the Cartesian product of two graphs $G\Box H$ is the graph with vertex set $V(G) \times V(H)$ and such that two vertices $(g_1, h_1), (g_2, h_2)$ are adjacent in $G\Box H$ if either $g_1 = g_2$ and h_1, h_2 are adjacent in H, or g_1, g_2 are adjacent in G and $h_1 = h_2$. We refer to [16] as a general reference about this topic. It is well known that domination-type parameters are difficult to handle in Cartesian product graphs and there is no general relationship between the value of such parameters in the product graph and its factor graphs. Even in the simplest cases of the Cartesian product of two graphs, that is, two paths

(grid), a path and a cycle (cylinder) and two cycles (torus) specific procedures are needed to compute such parameters.

The domination-type parameters in Cartesian product graphs are among the variety of graph parameters that can be computed by using matrix operations. This approach appeared for the first time in [4] and has been used in different Cartesian products, such as grids and cylinders, and also in different parameters, such as domination, independent domination and Roman domination (see for instance [17–21]). Unlike other parameters, those of domination-type do not use the usual matrix product but the $(\min, +)$ matrix product, which is also called the tropical product [1]. The $(\min, +)$ matrix product is defined over the semi-ring of tropical numbers $\mathcal{P} = (\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$ in the following way: $(A \boxtimes B)_{ij} = \min_k(a_{ik} + b_{kj})$. Moreover, for matrix A and $\alpha \in \mathbb{R} \cup \{\infty\}, (\alpha \boxtimes A)_{ij} = \alpha + a_{ij}$.

Graph algorithms involving tropical algebra operations can be found in the literature [22]. The computational side of this approach leads to interesting challenges bearing in mind the large size of the matrices involved in such algorithms and both, special properties of the matrices and regular structures of the graphs, have been taken into account in order to reduce the complexity of the matrix computations [23, 24]. Moreover, optimal implementations of the matrix operations in multicore and GPU platforms have proven to be suitable for these problems [25–27].

A contribution to the problem of the computation of the 2-domination number in cylinders can be found in [28], where this parameter was obtained in cylinders with a small cycle and any path, by using algorithms involving the $(\min, +)$ matrix-vector product. We now focus the complementary problem of computing this parameter in cylinders with a small path and any cycle, which is unknown. The technique we use here requires performing the $(\min, +)$ matrix-matrix product, which has higher computational requirements.

The goal of this work is twofold. From the computational point of view, efficient routines to compute $(\min, +)$ matrix products on multicore CPUs and GPUs are developed. Moreover, the matrices involved in the analysis of domination-type parameters in graphs are used to evaluate such implementations on modern HPC platforms. It is relevant to underline that, beyond this particular graph analysis, these efficient implementations are useful to accelerate the wide range of applications which are expressed in terms of $(\min, +)$ matrix products. To allow the scientific community to access to these efficient implementations of $(\min, +)$ matrix products, they are available at https://github.com/hpcjmart/2domination.

From the perspective of the graph analysis, our objective is to conjecture a formula for the 2-domination number in cylinders with path and cycle of unbounded order. Obtaining the value of the 2-domination number in cylinders with one small factor, either the path or the cycle, is the first step to addressing the general case. The reason is the regular behavior that is expected, except for the smallest cases. Making such regularity apparent provides the key information to look for the general formula.

In Sect. 2, we present the theoretical results that give support to the algorithms shown in Sect. 3 along with their computational analysis. Such algorithms will provide the desired values of the 2-domination number in cylinders with small path and any cycle, which we present in Sect. 4, as well as our conclusions from the computational point of view.

2 The 2-domination number in cylindrical graphs with small paths

In this section, we describe our approach to compute the 2-domination number of cylinders $P_m \square C_n$ with small paths. Such approach, involving the $(\min, +)$ matrix-matrix product has also been used to obtain similar results for the Roman domination number [20]. We first describe the general ideas involved in this method and then, we particularize the case of γ_2.

2.1 General construction

We focus on the following result from [29], that we quote from [4] in the version related to the $(\min, +)$ matrix product.

Let $\mathcal{D}$ be a digraph with vertex set $V(\mathcal{D}) = \{v_1, v_2, \ldots, v_s\}$ together with a labeling function ℓ which assigns an element of the semi-ring $\mathcal{P} = (\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$ to every arc of the digraph $\mathcal{D}$. A *path of length n* in $\mathcal{D}$ is a sequence of n consecutive arcs $Q = (v_{i_0} v_{i_1})(v_{i_1} v_{i_2}) \ldots (v_{i_{k-1}} v_{i_n})$ and Q is *a closed path* if $v_{i_0} = v_{i_n}$. The labeling ℓ can be easily extended to paths: $\ell(Q) = \ell(v_{i_0} v_{i_1}) + \ell(v_{i_1} v_{i_2}) + \cdots + \ell(v_{i_{k-1}} v_{i_n})$.

Theorem 1 [29] *Let S_{ij}^n be the set of all paths of length n from v_i to v_j in $\mathcal{D}$ and let $A(\mathcal{D})$ be the matrix defined by*

$$A(\mathcal{D})_{ij} = \begin{cases} \ell(v_i, v_j) & \text{if } (v_i, v_j) \text{ is an arc of } G, \\ \infty & \text{otherwise.} \end{cases}$$

If $A(\mathcal{D})^n$ is the n-th $(\min, +)$ power of $A(\mathcal{D})$, then $(A(\mathcal{D})^n)_{ij} = \min\{\ell(Q) : Q \in S_{ij}^n\}$.

The application of these results to the computation of domination-type parameters in Cartesian product graphs follows a common approach which uses the fact that these kinds of parameters are defined as the minimum cardinal of a set having a certain property. We now describe this general procedure.

Let G be a graph and let $a(G)$ be a parameter defined as the minimum cardinal of a vertex subset of G having a certain property A. First of all, we have to define a direct graph $\mathcal{D}$ such that there exists a bijective correspondence between the vertex subsets $U \subseteq V(G)$ having the property A and the closed paths Q of $\mathcal{D}$ with fixed length n, that we denote by $U \leftrightarrow Q$. As a second step, we have to define a labeling ℓ of the arcs of $\mathcal{D}$ such that if $U \leftrightarrow Q$ then, $|U| = \ell(Q)$. With such digraph and its associated labeling we can now use Theorem 1 to obtain $(A(\mathcal{D})^n)_{ii} = \min\{\ell(Q) : Q \in S_{ii}^n\} = \min\{|U|$

$:U \subseteq V(G)$ has property $A, U \leftrightarrow Q, Q \in S_{ii}^n\}$. That is, the $i - th$ entry $(A(\mathcal{D})^n)_{ii}$ of the main diagonal of the matrix $A(\mathcal{D})^n$ provides the minimum cardinal among all vertex subsets of G having property A and being identified with closed paths of $\mathcal{D}$ starting and ending in v_i. Finally, the minimum entry of the main diagonal of $A(\mathcal{D})^n$ gives the desired value of parameter $a(G)$:

$$\min_i (A(\mathcal{D})^n)_{ii} = \min_i (\min\{|U| : U \subseteq V(G) \text{ has property } A, U \leftrightarrow Q, Q \in S_{ii}^n\})$$
$$= \min\{|U| : U \subseteq V(G) \text{ has property } A\} = a(G)$$

A restriction that occurs when using this approach to compute a parameter $a(G)$ is that graph G needs some structure that allows us to identify the vertex subsets $U \subseteq V(G)$ having the property A and the closed paths Q of $\mathcal{D}$ with fixed length n. The Cartesian products of paths and cycles have such structure, as we now briefly sketch. The cylinder $P_m \square C_n$ has vertex set $V(P_m \square C_n) = \{u_{ij} : 0 \leq i \leq m - 1, 0 \leq j \leq n - 1\}$. The $j - th$ column is the subgraph generated by $\{u_{ij} : 0 \leq i \leq m - 1\}$, which is isomorphic to P_m.

Let $U \subseteq V(P_m \square C_n)$ be a vertex subset having the property A and let us consider U_j the $j - th$ column of $P_m \square C_n$, taking into account whether or not its vertices belong to U (by using a labeling of the vertices). The vertices of the digraph $\mathcal{D}$ are all possible U_j obtained in such way, for every vertex subset having property A. Moreover, there is an arc from U_r to U_{r+1}, that is, there is an arc from a vertex of $\mathcal{D}$ to another one if they are consecutive columns in $P_m \square C_n$ for the same vertex subset U having property A. Then, U can be identified with the closed path $Q = (U_1, U_2), (U_2, U_3) \ldots (U_n, U_1)$ that has fixed length n.

The key point of the construction above is the column structure of the cylinder $P_m \square C_n$ and additional requirements are needed in such construction depending on the studied parameter $a(G)$. In this paper we focus on 2-domination number γ_2 of the cylinder $P_m \square C_n$ and a suitable digraph $\mathcal{D}$ will be defined. The $(\min, +)$ powers of the matrix $A(\mathcal{D})$ have to be computed and this matrix is expected to be quite large, to such an extent as digraph $\mathcal{D}$ is much larger than the cylinder $P_m \square C_n$. Indeed, the matrix size exponentially grows with the order of the cylinder and for this reason, this approach is useful just in cylinders $P_m \square C_n$ with small enough values of both m and n. An additional procedure involving well-known properties of the $(\min, +)$ matrix product allows the removal of one of such size restrictions.

2.2 Specific construction for the 2-domination number

Let $P_m \square C_n$ be a cylinder and let $S \subseteq V(P_m \square C_n)$ a 2-dominating set. We label the vertices in the cylinder according to the following rules:

- $v = 0$ if $v \in S$,
- $v = 1$ if $v \notin S$ and v has at least 2 neighbors in S in its column or the previous one,
- $v = 2$ if $v \notin S$ and v has just 1 neighbor in S in its column or the previous one.

We now identify each column with a word $p = (p^1, p^2, \ldots, p^m)$ with length m in the alphabet $\{0, 1, 2\}$ and containing neither the sequences 020, 111, 211, 112, 212 in any position, nor the sequences 11, 12 at the beginning (that is, for the letters $p^1 p^2$) nor the sequences 11, 21 at the end (that is, for the letters $p^{m-1} p^m$). These restrictions come from the fact that S is a 2-dominating set and from the definition of the labeling. We call *correct m-words* to words of length m in the alphabet $\{0, 1, 2\}$

fulfilling all the conditions above. We define the vertex set of the digraph $\mathcal{D}_m$ as the set of all correct m-words.

We now focus on the definition of the arcs in the digraph $\mathcal{D}_m$. Given two correct m-words $p = (p^1, p^2, \dots, p^m)$ and $q = (q^1, q^2, \dots, q^m)$, we say that p can follow a q if they can be consecutive columns (in the order qp) in some 2-dominating set, that is, they follow the rules of the labeling:

- if $q_i = 2$ then $p_i = 0$,
- if $p_i = 2$ then exactly one among p_{i-1}, p_{i+1}, q_i is equal to 0 (if $i = 1$ then exactly one among p_{i+1}, q_i is equal to 0 and if $i = m$ then exactly one among p_{i-1}, q_i is equal to 0),
- if $p_i = 1$ then at least two among p_{i-1}, p_{i+1}, q_i is equal to 0 (the same comment as above for cases $i = 1$ and $i = m$).

Finally, there is an arc from a word q to a word p if and only if p can follow q. This concludes the construction of the digraph $\mathcal{D}_m$, and it is clear that every 2-dominating set S of $P_m \square C_n$ is univocally identified with a closed path Q of length n, that is, $S \leftrightarrow Q$.

We now need to define a labeling of the arcs of $\mathcal{D}_m$ fulfilling that if $S \leftrightarrow Q$ then, $|S| = \ell(Q)$. To this end, for an arc (q, p) we define its label as $\ell(q, p) =$ number of zeros of p, which obviously gives the desired property. We illustrate the definitions above with an example.

Example 1 In Fig. 1 a 2-dominating set of $P_4 \square C_5$ is shown (black vertices). Moreover, the list of correct words representing the columns of such 2-dominating sets are in Fig. 1.

Clearly p_{i+1} can follow p_i for $i \in \{1, 2, 3, 4\}$ and p_1 can follow p_5 so $Q = (p_1, p_2), (p_2, p_3), (p_3, p_4), (p_4, p_5), (p_5, p_1)$ is a closed path in the digraph $\mathcal{D}_4$. The label of each arc of Q is the number of zeros in the second word, that is, $\ell(p_1, p_2) = 2, \ell(p_2, p_3) = 2, \ell(p_3, p_4) = 2, \ell(p_4, p_5) = 1, \ell(p_5, p_1) = 3$. Hence $\ell(Q) = 2 + 2 + 2 + 1 + 3 = 10$, that reflects that the 2-dominating set has 10 vertices.
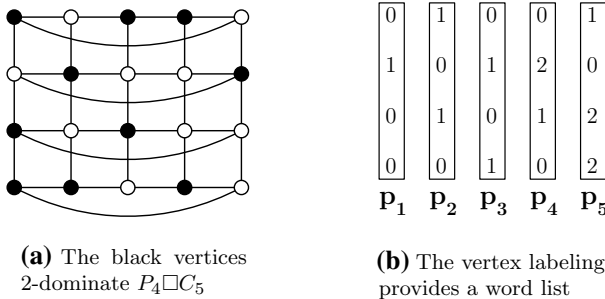

(a) The black vertices 2-dominate $P_4 \square C_5$

(b) The vertex labeling provides a word list

Fig. 1 A 2-dominating set of $P_4 \square C_5$ and its associated word list

Theorem 2 *Let $P_m \square C_n$ be a cylinder and let $\mathcal{D}_m$ be the digraph constructed above, with the arc labeling ℓ. Let S_{qp}^n be the set of all paths of length n from q to p in $\mathcal{D}_m$ and let $A(\mathcal{D}_m)$ be the matrix defined by*

$$A(\mathcal{D}_m)_{qp} = \begin{cases} \ell(q,p) & \text{if } (q,p) \text{ is an arc of } G, \\ \infty & \text{otherwise.} \end{cases}$$

If $A(\mathcal{D}_m)^n$ is the (min, +) *power of $A(\mathcal{D}_m)$ then, $\min_i(A(\mathcal{D}_m)^n)_{ii} = \gamma_2(P_m \square C_n)$.*

Proof The proof comes from Theorem 1 and the specific constructions of the digraph $\mathcal{D}_m$ and the labeling ℓ. □

Roughly speaking, Theorem 1 says that the entry (i, j) of the matrix $A(\mathcal{D}_m)^n$ gives the minimum label among all paths in $\mathcal{D}_m$ with length n, beginning in p_i and ending in p_j. Therefore, the entry (i, i) on the main diagonal shows the minimum label among all closed n-paths that begin and end in p_i. Each closed path represents a 2-dominating set of $P_m \square C_n$ and its label is the cardinal of such set (see Fig. 1). Hence, Theorem 2 says that the minimum entry of the main diagonal gives the minimum cardinal among all 2-dominating sets, that is, the 2-dominating number.

Using Theorem 2 to compute the 2-domination number of $P_m \square C_n$ is subject to certain restrictions for both m and n. On the one hand, the path order m determines the number of correct m-words and therefore, the size of the matrix $A(\mathcal{D}_m)$ that is expected to be of the order of 3^m. On the other hand, the cycle order n is the number of (min, +) matrix powers that have to be computed to obtain the value of the 2-domination number. The first limitation is intrinsic to this approach. However, there are some properties of the (min, +) matrix product that can avoid the second one.

Lemma 1 *Let M be a square matrix. Suppose that there exist natural numbers n_0, a, b such that $M^{n_0+a} = b \boxtimes M^{n_0}$. Then, $M^{n+a} = b \boxtimes M^n$, for every $n \geq n_0$.*

Proof By hypothesis, $M^{n_0+a} = b \boxtimes M^{n_0}$. Let $n \geq n_0$ be such that $M^{n+a} = b \boxtimes M^n$ then, $M^{(n+1)+a} = M \boxtimes M^{n+a} = M \boxtimes (b \boxtimes M^n) = b \boxtimes (M \boxtimes M^n) = b \boxtimes M^{n+1}$. □

Theorem 3 *Let $m \geq 2$ be an integer and suppose that there exist natural numbers n_0, a, b such that $A(\mathcal{D}_m)^{n_0+a} = b \boxtimes A(\mathcal{D}_m)^{n_0}$. Then, the 2-domination number satisfies the finite difference equation $\gamma_2(P_m \square C_{n+a}) - \gamma_2(P_m \square C_n) = b, n \geq n_0$.*

Proof By Lemma 1, we know that $A(\mathcal{D}_m)^{n+a} = b \boxtimes A(\mathcal{D}_m)^n$ for every $n \geq n_0$. Now, by Theorem 2 we obtain $\gamma_2(P_m \square C_{n+a}) = \min_i(A(\mathcal{D}_m)^{n+a})_{ii} = \min_i(b \boxtimes A(\mathcal{D}_m)^n)_{ii} = b + \min_i(A(\mathcal{D}_m)^n)_{ii} = b + \gamma_2(P_m \square C_n)$, for $n \geq n_0$. □

Assuming that m is small enough to apply Theorem 2 and that n_0, a, b have been obtained for m then, the boundary values of the finite difference equation above, that is, $\gamma_2(P_m \square C_n)$ for $n_0 \leq n \leq n_0 + a - 1$ can be computed by using Theorem 2 and the finite difference equation can be easily solved to obtain the formula for

the 2-domination number $\gamma_2(P_m\square C_n)$, for $n \geq n_0$. Moreover, the remaining values $\gamma_2(P_m\square C_n)$ for $n < n_0$, if any, can also be computed by Theorem 2. Thus, if m is small enough to apply Theorem 2 and the conditions of Theorem 3 hold, then $\gamma_2(P_m\square C_n)$ can be obtained for any $n \geq 3$.

3 Algorithms and computational analysis

In this section, we present the algorithms we have used to compute the 2-domination number of $P_m\square C_n$, with $2 \leq m \leq 12$ and $n \geq 3$. We also study the performance of such algorithms in sequential and parallel implementations on a CPU AMD EPYC Rome 7642 with 48 cores and, in addition, on a GPU NVIDIA Tesla V100-PCIE with 32 GB of memory, 80 multiprocessors with 128 cores in each multiprocessor (10240 cores CUDA).

Algorithms from 1 to 4 come from Theorem 3 and they allow us to pose the finite difference equation involving the 2-domination number of $P_m\square C_n$, with m small enough. Moreover, Theorem 2 provides Algorithm 5 to compute the boundary values of the finite difference equations. Our first target is to obtain the suitable values a_m, b_m, n_0^m to pose such equation for each $m \in \{2, \ldots, 12\}$ and first of all, we compute the matrix $A(\mathcal{D}_m)$ in Algorithm 1. In order to obtain the set $\mathcal{C}_m$ of all correct m-words, we first obtain all the m-element variations of 3-elements 0, 1, 2, with repetition allowed. Then, we select those of them not containing the forbidden sequences of the correct m-words.

Algorithm 1 is only useful for small values of m. As we said before, the size of the matrix $A(\mathcal{D}_m)$ is expected to exponentially grow with m, as do the necessary computational resources to get and manage such matrix.

Algorithm 1 Computation of matrix $A(\mathcal{D}_m)$

Require: $m \geq 2$, $\mathcal{C}_m$
Ensure: matrix $A(\mathcal{D}_m)$
  1: **for** each $q_i \in \mathcal{C}_m$ **do** $\triangleright\ i = 1;\ i \leq |\mathcal{C}_m|;\ i{+}{+}$
  2:     **for** each $p_j \in \mathcal{C}_m$ **do** $\triangleright\ j = 1;\ j \leq |\mathcal{C}_m|;\ j{+}{+}$
  3:         condition=check p_j can follow q_i;
  4:         **if** condition==True **then**
  5:             $A(\mathcal{D}_m)_{(i,j)}$ =number of zeros of p_j;
  6:         **else**
  7:             $A(\mathcal{D}_m)_{(i,j)} = \infty$;
  8:         **end if**
  9:     **end for**;
10: **end for**;

In Table 1 we show the matrix sizes and the memory requirements in cases $2 \leq m \leq 13$, by using 16 bits arithmetic types of integers. The memory size of the matrix in the case $m = 13$ makes it unfeasible to allocate it into the GPU memory,

Table 1 Size of the matrix $A(\mathcal{D}_m)$ in Algorithm 1

m	Rows	Memory size	m	Rows	Memory size
2	6	0.07 KB	8	1386	3.67 MB
3	15	0.45 KB	9	3447	22.67 MB
4	36	2.53 KB	10	8568	140.02 MB
5	90	15.82 KB	11	21294	0.85 GB
6	225	98.88 KB	12	52929	5.22 GB
7	558	0.69 MB	13	131562	32.24 GB

Table 2 Running times of Algorithm 2 to compute $A(\mathcal{D}_m)^k, k \leq 50$

m	Sequential	Multicore 48 threads		GPU	
	Time	Time	Sequ./multicore speedup	Time	Multicore/ GPU speedup
7	13.4s	0.4s	33.5	0.2s	2
8	3 m 18.9 s	5.3 s	37.5	0.5s	10.6
9	56 m 42.3 s	1 m 31.8 s	37.1	2.9s	31.7
10	17 h 9 m 21.6 s	25 m 23.4 s	40.5	30.2 s	50.4
11	–	6 h 29 m 28.5 s		6 m 21.6 s	61.2
12	–	–		1 h 30 m 15.6 s	

which is the processor we have used to accelerate our algorithms. This is the reason we have analyzed, in this paper, the cases $2 \leq m \leq 12$. We have run Algorithm 1 in the CPU and it takes 2 minutes in the larger case $m = 12$. This running time is small compared with the following algorithms and moreover, the algorithm does not use any matrix operations whose analysis is our objective. Therefore, we have not parallelized this process and the matrix $A(\mathcal{D}_m)$ is an input data for the remaining algorithms.

We now need enough $(\min, +)$ powers of the matrix $A(\mathcal{D}_m)$ in order to look for the recurrence relationship. We obtain the desired powers with Algorithm 2.

Algorithm 2 Computation of K $(\min, +)$ powers of $A(\mathcal{D}_m)$

Require: $m \geq 2$, $A(\mathcal{D}_m)$ and $K \geq 2$
Ensure: K $(\min, +)$ powers of $A(\mathcal{D}_m)$
  1: initialize $A(\mathcal{D}_m)$; ▷ Read file from disk
  2: **for** $i = 1; i <= K; i ++$ **do**
  3:    $A(\mathcal{D}_m)^i = A(\mathcal{D}_m) \boxtimes A(\mathcal{D}_m)^{i-1}$; ▷ $(\min, +)$ Matrix Product
  4: **end for**;

Table 3 Results obtained by Algorithm 3

m	r_0^m	a_m	b_m	m	r_0^m	a_m	b_m
2	48	2	2	8	47	3	10
3	44	6	8	9	47	3	11
4	42	8	14	10	47	3	12
5	43	7	15	11	47	3	13
6	39	11	28	12	47	3	14
7	32	18	53				

There exist sufficient but not necessary conditions ensuring that the hypotheses in Theorem 3 are true (see [30]). However, such conditions provide a non-minimum value for n_0 in the order of the square of the matrix size that is not practical. We have run Algorithm 2 with $K = 50$, which has proven to be enough in cases $2 \leq m \leq 12$.

Due to the high requirements to sequentially compute the powers, we have modified this routine in two ways to accelerate it on modern multicore CPU and GPUs. On the one hand, we have used the directives of OpenMP [31] to parallelize the $(\min, +)$ matrix multiplication on multicore CPUs. Specifically, we use the OpenMP directives to accelerate the computation of each product, so the outer loop that iterates through the rows of the first matrix of the product is parallelized. This technique is straightforward, and it allows to efficiently develop the $(\min, +)$ matrix product to leverage the resources of the CPU multicore processors. Moreover, the performance achieved is enough for the purpose of our work when the dimensions of the matrices are moderated.

On the other hand, the powers have also been carried out by a modification of the routine MatrixMul, available in the NVIDIA CUDA TOOLKIT 11 [32] and described in the CUDA C Programming Guide (see [33], Chapter 3), to adapt it to the $(\min, +)$ multiplication. In this case, we use a different parallelization strategy than the one used in OpenMP. It is based on a tiled matrix multiplication to optimize the GPU hierarchy memory management. So, this method takes advantage of the lower latency, the higher bandwidth shared memory within GPU thread blocks and the number of slow accesses to memory device, which are minimized. For details of the memory access pattern of MatrixMul see Chapter 3 of [33].

We show in Table 2 the running times of Algorithm 2 in cases $7 \leq m \leq 12$ while in the remaining cases the algorithm needs less than 1 second, even with the sequential implementation.

Table 2 shows that the running time of computing 50 $(\min, +)$ powers of matrix $A(\mathcal{D}_m)$ exponentially grows as the matrix size increases. In order to address large cases in reasonable time we have run an OpenMP parallel implementation with 48 cores/threads. Such implementation provides small running times in cases $m = 8$ and $m = 9$ but it grows fast for $m \geq 10$. In order to increase the efficiency of this algorithm, we have run a version of the $(\min, +)$ matrix product in CUDA for NVIDIA GPU and we have obtained a significant improvement in terms of running times compared to the sequential and the parallel OpenMP versions.

The following step to apply Theorem 3 is to find the appropriate recurrence relationship between two powers of matrix $A(\mathcal{D}_m)$. Even though such matrix is sparse, we have noted that its powers become dense, that is, with no infinite entries, from the third one. Therefore, the hypothesis in Theorem 3, that is, $A(\mathcal{D}_m)^{n_0+a} = b \boxtimes A(\mathcal{D}_m)^{n_0}$ is equivalent to $A(\mathcal{D}_m)^{n_0+a} - A(\mathcal{D}_m)^{n_0}$ being a constant matrix with entries equal to b_m. We use this fact in Algorithm 3. The results are shown in Table 3 ($K = 50$).

Table 4 Values to apply Theorem 3 obtained with Algorithm 4

m	n_0^m	a_m	b_m	m	n_0^m	a_m	b_m
2	4	2	2	8	25	3	10
3	7	6	8	9	22	3	11
4	9	8	14	10	21	3	12
5	31	7	15	11	24	3	13
6	19	11	28	12	26	3	14
7	23	18	53				

Algorithm 3 Search for the recurrence relation

Require: $m \geq 2$ and K (min, +) powers of $A(\mathcal{D}_m)$
Ensure: a_m, b_m, r_0^m
1: **for** $i = K$; $i >= 1$; $i - -$ **do**
2:     $A(\mathcal{D}_m)^i$;
3:     **for** $j = i - 1$; $j >= 1$; $j - -$ **do**
4:         initialize $A(\mathcal{D}_m)^j$;
5:         **if** $(A(\mathcal{D}_m)^i - A(\mathcal{D}_m)^j) == constant$ **then**
6:             $a_m = i - j$;   ▷ Steep
7:             $b_m = constant$ ▷ Difference value
8:             $r_0^m = j$;
9:             Exit;
10:        **end if**;
11:    **end for**;
12: **end for**;

It is expected that the values of r_0^m are not minimum because we have found a recurrence relationship with $r_0^m + a_m = 50$, for every m. But in any case, we have confirmed that matrix $A(\mathcal{D}_m)$ meets the hypothesis of Theorem 3 and the finite difference equation can be posed for $n \geq r_0^m$.

We now show how to obtain the minimum value n_0^m such that $A(\mathcal{D}_m)^{n+a_m} = b_m \boxtimes A(\mathcal{D}_m)^n$ for every $n \geq n_0^m$, in Algorithm 4 . Finding this optimal value could be interesting in order to try to reduce the number of (min, +) powers required to ensure the hypothesis of Theorem 3.

Algorithm 4 Search for the minimum values of the recurrence relationship

---

Require: $m \geq 2$ and r_0^m, a_m, b_m
Ensure: n_0^m
  1: **for** $i = (r_0^m + a_m); i >= 1; i--$ **do**
  2:     **if** $(A(\mathcal{D}_m)^i - A(\mathcal{D}_m)^{i-a_m}) \neq b_m$ **then**
  3:         Exit;
  4:     **else**
  5:         $n_0^m = i - a_m;$
  6:     **end if**;
  7: **end for**;

We show the values of n_0^m obtained with Algorithm 4 in Table 4, together with the values of a_m, b_m shown before. Such values provide the finite difference equation $\gamma_2(P_m \Box C_{n+a_m}) - \gamma_2(P_m \Box C_n) = b_m, n \geq n_0^m$ and $m \in \{2, \dots, 12\}$.

The matrix operation used in Algorithms 3 and 4 is the matrix difference, which consumes fewer computational resources than the $(\min, +)$ matrix multiplication. Indeed, both algorithms are faster with the OpenMP directives than on the GPU due to the cost of communications to allocate the matrices on the GPU memory to perform quite a simple operation. For instance, the running times (in seconds) of Algorithm 3 for largest case we have computed $m = 12$ are 16.8 on the CPU (sequential), 13.6 on the GPU and 7.2 with OpenMP (48 cores). For Algorithm 4, they are 149.8, 170.0 and 98.5, respectively.

Finally, we compute the boundary values needed to solve the finite difference equations and to obtain the formulæ of the 2-domination number in the studied cases, with Algorithm 5, by using Theorem 2.

Algorithm 5 $\gamma_2(P_m \Box C_n)$, for $3 \leq n \leq n_0^m + a_m - 1$

Require: $m \geq 2$ and $(n_0^m + a_m - 1)$ $(\min, +)$ powers of $A(\mathcal{D}_m)$
Ensure: $\gamma_2(P_m \Box C_n), 3 \leq n \leq (n_0^m + a_m - 1)$
  1: **for** $i = 3; i <= (n_0^m + a_m - 1); i++$ **do**
  2:     $\min(\text{main diagonal}(A(\mathcal{D}_m)^i))$
  3: **end for**

Algorithm 5 uses the minimization operation over the main diagonal of the matrix $A(\mathcal{D}_m)^i$, which can be seen as a vector with a length of the number of rows of the matrix. This matrix operation is less computationally demanding given that the number of the operations needed here is on the order of the number of rows of the matrix while in Algorithms 3 and 4 the order is the square of that number. Indeed, the CPU needs less than 1 second if $m \leq 11$ and 11.8 seconds in the largest case $m = 12$. Our program to compute the 2-domination number of cylindrical graphs with small paths consists of consecutive run Algorithms from 2 to 5 and we have implemented it in four ways. The first one runs every algorithm on the CPU and we

Table 5 Total running times

m	Version 1	Version 2	Version 3	Version 4
7	13.5 s	0.4 s	0.2 s	0.3 s
8	3 m 19 s	5.4 s	0.6 s	0.6 s
9	56 m 43 s	1 m 32 s	3.9 s	3.5 s
10	17 h 9 m 27 s	25 m 26 s	35 s	32.5 s
11		6 h 29 m 45 s	6 m 44 s	6m 32 s
12			1 h 33 m 34 s	1 h 29 m 58 s

have here completed the computation of cases $m \leq 10$, due to high running times of Algorithm 2.

In the second version, we have used the OpenMP directives to parallelize the execution of the (min, +) matrix product routines in Algorithm 2 and the matrix difference in Algorithm 3 and 4 because they are the most computationally demanding matrix operations. We have computed until case $m = 11$ with 48 cores and although the speedup for Algorithm 2 is over 40 in the last case, the running time is still huge. The third program runs Algorithms 2, 3 and 4 on the GPU and cases $m \leq 12$ have been obtained. Algorithm 2 presents here a very noticeable improvement in terms of running time, but the huge matrix size does not allow us to approach large cases given that from $m = 13$ the matrix cannot be allocated on the GPU memory.

In order to test the goodness of the implementation of Algorithms 3 and 4 on the CPU compared to the GPU, we have done the fourth version that uses the GPU just in Algorithm 2 and the OpenMP parallelization for Algorithms 3 and 4. This is slightly faster than version 3 because of the communication costs to allocate matrices on the GPU memory to perform matrix operations with little computational cost. The total running times of the four versions are shown in Table 5.

Table 6 Values of α_k^m

m	a_m	k with $\alpha_k^m = 1$	k with $\alpha_k^m = 0$
2	2	None	All
3	6	None	All
4	8	4, 5	Otherwise
5	7	None	All
6	11	5, 9	Otherwise
7	18	$k \in \{1, 2, 4, 5\}$, $19 \leq n \equiv k \pmod{18}$	Otherwise
8*	3	None	All
9	3	None	All
10*	3	1, 2	Otherwise
11	3	2	Otherwise
12*	3	1, 2	Otherwise

* There is one exception

4 Conclusions

According to Theorem 3, values in Table 4 allow us to pose the finite difference equation $\gamma_2(P_m \square C_{n+a_m}) - \gamma_2(P_m \square C_n) = b_m, n \geq n_0^m$, for each $2 \leq m \leq 12$. The boundary values $\gamma_2(P_m \square C_n), n_0^m \leq n \leq n_0^m + a_m - 1$, have been obtained with Algorithm 5. Therefore, the solution is $\gamma_2(P_m \square C_n) = \left\lceil \frac{b_m \cdot n}{a_n} \right\rceil + \alpha_k^m$, where $n \equiv k \pmod{a_m}$ and α_k^m depends on the boundary values for each m. Moreover, the remaining values of $\gamma_2(P_m \square C_n)$, for $3 \leq n < n_0^m$, have also been computed with Algorithm 5 and most of them follow the general formula.

In the same way as in other domination parameters in grids and cylinders (see [17, 18]), these results show a non-regular behavior for the smallest values of m, but it becomes regular for $m \geq 8$. Note that if $8 \leq m \leq 12$ then, $a_m = 3$ and $b_m = m + 2$. In such cases $\gamma_2(P_m \square C_n) = \lceil \frac{(m+2)n}{3} \rceil + \alpha_k^m$, where $n \equiv k \pmod 3$ and α_k^m again depends on the boundary values $\gamma_2(P_m \square C_{n_0^m + k})$. In order to complete the formulæ, in Table 6 we show the values of α_k^m, for each $m \in \{2, \ldots, 12\}$ and $k \in \{0, \ldots, a_m - 1\}$.

The only exceptions are $n = 5$, for $m \in \{8, 10, 12\}$, where $\alpha_k^m = 2$. This value is coherent with the results obtained in [28]: $\gamma_2(C_5 \square P_m) = 2m + 2$ if $2 < m \equiv 0 \pmod 2$ and $\gamma_2(C_5 \square P_m) = 2m + 1$ if $m = 2$ or $m \equiv 1 \pmod 2$.

In spite of obtaining that $\alpha_k^m \leq 2$ for $m \leq 12$, we think that such numbers will increase for some values of n as m grows because they would depend on m in some way. Our results cover the cases $2 \leq m \leq 12$, $3 \leq n \leq 15$ already studied in [28], and all the results match. In addition, for $8 \leq m \leq 12$ and $n \equiv 0 \pmod 3$ we have shown that $\gamma_2(P_m \square C_n) = \frac{(m+2)n}{3}$. The same formula for $n = 3, 6, 9, 12, 15$ and $m \geq 8$ is obtained in [28] and we have now extended this result to every $n \equiv 0 \pmod 3$, for $8 \leq m \leq 12$. Also note that our formulæ for $m \leq 7$ and $n \equiv 0 \pmod 3$ show that such small cases do not follow the same formula, in general. Our results together with those in [28] give us support to conjecture that $\gamma_2(P_m \square C_n) = \frac{(m+2)n}{3}$, if $m \geq 8$ and $n \equiv 0 \pmod 3$.

Regarding the computational point of view, our main target was to develop efficient routines to compute $(\min, +)$ matrix products on multicore CPUs and GPUs. Such routines have application to the computation of the 2-domination number of cylindrical graphs with small paths of order m. Our approach has as a limitation the size of the involved matrices that exponentially grows as m does. This condition has led us to focus on cases $2 \leq m \leq 12$ that meet the requirements of our computational resources on both the CPU and the GPU.

Once we have obtained the matrices for cases $2 \leq m \leq 12$, we have divided the routines in Algorithms from 2 to 5 and three of them, Algorithms 3, 4 and 5, can be run on the CPU in a reasonable time. Moreover, the OpenMP parallelization with 48 cores slightly improves such running times, which are negligible compared to the total ones. However, the CPU has shown to be non-sufficient to run Algorithm 2 in the most interesting cases, which are the largest ones, to find the desired regular behavior of the 2-domination number. The matrix operation used by this algorithm is the $(\min, +)$ matrix product and we explore two improvement options to reduce its running time: a parallelization of the algorithm with OpenMP with 48

cores and an implementation of this matrix product in CUDA for NVIDIA GPU. The OpenMP parallel version with 48 cores of Algorithm 2 has shown a speedup over 40 regarding the sequential version in case $m = 10$. However, the running times are so high that the parallelization is not enough for $m \geq 11$, where more than 6 hours are needed. In contrast, the GPU version computes 50 powers of the matrix $A(\mathcal{D}_m)$ in considerably less time, with a speedup over 60 compared to the OpenMP version for $m = 12$.

We think it would be possible to improve the efficiency of Algorithm 2 by reducing the number of computed powers while the finite difference equation can still be solved. In addition, some parallelization of the (min, +) product allowing to distribute the product of two matrices in small sets of rows and columns would give the opportunity of computing some cases larger than $m = 12$. Such improvements would perhaps allow us to conjecture a general formula of the 2-domination number of the cylinder $P_m \square C_n$ with $n \equiv 1, 2 \pmod 3$.

To sum up, we have solved the graph problem of computing the 2-domination number of some cylinders with a small path in a reasonable time by exploiting the benefits of the GPU's to run algorithms involving the (min, +) matrix product while the rest of matrix operations involved, such as the matrix difference or the minimization of the main diagonal of a matrix, demand fewer computational resources and they can be addressed on the multicore CPU in a short time. Finally, we have conjectured that $\gamma_2(P_m \square C_n)$ if $n \equiv 0 \pmod 3$.

Data Availability Statement The source code, in programming language C, of the algorithms developed in this paper can be found online in the repository https://github.com/hpcjmart/2domination.

References

1. Pin J-E (1998) Tropical semirings, Idempotency. In: Gunawardena J (ed) Publications of the Newton Institute. Cambridge University Press, Cambridge, UK, pp 50–69. https://doi.org/10.1017/CBO9780511662508.004
2. Omanovic A, Kazan H, Oblak P, Curk T (2021) Sparse data embedding and prediction by tropical matrix factorization. BMC Bioinform 22(1):89

3. Speyer D, Sturmfels B (2009) Tropical mathematics. Math Mag 82(3):163–173. https://doi.org/10.1080/0025570X.2009.11953615

4. Klavžar S, Žerovnik J (1996) Algebraic approach to fasciagraphs and rotagraphs. Discret Appl Math 68(1):93–100. https://doi.org/10.1016/0166-218X(95)00058-Y

5. Butkovič P (2019) A note on tropical linear and integer programs. J Optim Theory Appl 180(3):1011–1026. https://doi.org/10.1007/s10957-018-1429-8

6. Krivulin N (2015) Algebraic solutions of tropical optimization problems. Lobachevskii J Math 36(4):363–374. https://doi.org/10.1134/S199508021504006X

7. Haynes TW, Hedetniemi ST, Slater PJ (1998) Fundamentals of domination in graphs. Chapman and hall CRC pure and applied mathematics series, Marcel Dekker Inc, New York, USA

8. Fink JF, Jacobson MS (1985) N-domination in graphs. Graph theory with applications to algorithms and computer science. Wiley, USA, pp 283–300

9. Bujtás C, Jaskó S (2018) Bounds on the 2-domination number. Discrete Appl Math 242:4–15. https://doi.org/10.1016/j.dam.2017.05.014

10. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman, New York, USA

11. Jacobson MS, Peters K (1989) Complexity questions for n-domination and related parameters. Congr Numer 68:7–22

12. Bean TJ, Henning M, Swart HC (1994) On the integrity of distance domination in graphs. Australas J Comb 10:29–44

13. Vizing VG (1968) Some unsolved problems in graph theory. Uspekhi Mat Nauk 23(6):117–134

14. Brešar B, Dorbec P, Goddard W, Hartnell BL, Henning MA, Klavžar S, Rall DF (2012) Vizing's conjecture: a survey and recent results. J Graph Theory 69(1):46–76. https://doi.org/10.1002/jgt.20565

15. Brešar B, Hartnell BL, Henning MA, Kuenzel K, Rall DF (2021) A new framework to approach Vizing's conjecture. Discuss Math Graph Theory 41(3):749–762. https://doi.org/10.7151/dmgt.2293

16. Imrich W, Klavžar S (2000) Product Graphs, Structure and Recognition. In: Wiley-Interscience series in discrete mathematics and optimization, Wiley, New York. p 358

17. Crevals S (2014) Domination of cylinder graphs. Congr Numer 219:53–63

18. Gonçalves D, Pinlou A, Rao M, Thomassé S (2011) The domination number of grids. SIAM J Discret Math 25(3):1443–1453. https://doi.org/10.1137/11082574

19. Guichard DR (2004) A lower bound for the domination number of complete grid graphs. J Combin Math Combin Comput 49:215–220

20. Martínez JA, Garzón EM, Puertas ML (2021) Powers of large matrices on GPU platforms to compute the roman domination number of cylindrical graphs. IEEE Access 9:29346–29355. https://doi.org/10.1109/ACCESS.2021.3058738

21. Pavlič P, Žerovnik J (2012) Roman domination number of the cartesian products of paths and cycles. Electron J Comb 19(3):19

22. Kepner J, Gilbert JR (eds.): Graph Algorithms in the Language of Linear Algebra. Software, environments, tools, vol. 22. SIAM, Philadelphia, USA (2011). https://doi.org/10.1137/1.9780898719918

23. Dobosiewicz W (1990) A more efficient algorithm for the min-plus multiplication. Int J Comput Math 32(1–2):49–60. https://doi.org/10.1080/00207169008803814

24. Felzenszwalb PF, McAuley JJ (2011) Fast inference with min-sum matrix product. IEEE Trans Pattern Anal Mach Intell 33(12):2549–2554. https://doi.org/10.1109/TPAMI.2011.121

25. Buluç A, Gilbert J (2011) The combinatorialBLAS: design, implementation, and applications. Int J High Perform Comput Appl 25:496–509. https://doi.org/10.1177/1094342011403516

26. Humayun A, Asif M, Hanif MK (2017) BTAS: A library for tropical algebra. CoRR **abs/1701.04733**

27. Yang C, Buluç A, Owens JD (2019) Graphblast: a high-performance linear algebra-based graph framework on the GPU. CoRR **abs/1908.01407**

28. Garzón EM, Martínez JA, Moreno JJ, Puertas ML (2022) On the 2-domination number of cylinders with small cycles. Fund. Inform. **accepted**

29. Carré B (1979) Graphs and Networks. Clarendon Press, Oxford, UK

30. Spalding A (1998) Min-plus algebra and graph domination. PhD thesis, Dept. of Appl. Math., Univ. of Colorado, Denver, CL, USA

31. TheOpenMP API specification for parallel programming. https://www.openmp.org. Accessed: 2021-03-31

32. NVIDIA CUDA toolkit. https://developer.nvidia.com/cuda-math-library. Accessed: 2021-03-31
33. NVIDIA CUDA documentation. https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf. Accessed: 2021-03-31