Communications in Combinatorics and Optimization
Vol. xx, No. x (xxxx), pp. 1-13
https://doi.org/10.22049/cco.2026.31431.2848

CCO
Commun. Comb. Optim.

Research Article

Algorithmic aspects of independent dominating set in grid graphs

Hamid Gholami[1,†], Doost Ali Mojdeh[1,*], Mahdi Imanparast[2]

[1]Department of Mathematics, Faculty of Mathematical Sciences,
University of Mazandaran, Babolsar, Iran
[†]gholami.hamid2005@gmail.com
[*]damojdeh@umz.ac.ir

[2]Department of Computer Science, Faculty of Basic Sciences,
University of Bojnord, Bojnord, Iran
m.imanparast@ub.ac.ir

Received: 1 February 2026; Accepted: 2 November 2026
Published Online: 12 November 2026

Abstract: For a graph $G(V, E)$, a dominating set D is defined as a subset $D \subseteq V$ such that every vertex in the graph is either included in D or is adjacent to at least one vertex in D. A set $D \subseteq V$ of vertices is called an independent dominating set if D is a dominating set and no two vertices in D are adjacent. In this study, we present a 2-approximation algorithm for computing an independent dominating set of grid graphs with a running time $O(n^2)$. Our proposed algorithm employs a greedy selection strategy that iteratively identifies vertices based on their potential to cover the maximum number of uncovered neighbors. We establish that our algorithm guarantees an approximation factor of 2, meaning the number of vertices in the independent dominating set produced by the algorithm is at most twice the number of vertices in an optimal solution.

Keywords: Approximation algorithms, NP-hard problems, grid graphs, dominating sets, independent dominating sets.

AMS Subject classification: 05C85, 68W25, 68Wxx

1. Introduction

The study on the family of dominating sets in graph theory and designing algorithms for them [19, 21–24] has garnered significant attention due to its wide-ranging applicability across various domains, including computer networks, resource allocation,

* *Corresponding Author*

© 2026 Azarbaijan Shahid Madani University

social networks, urban planning, and sensor coverage. Let an optimization problem be given, if there is no efficient polynomial algorithm to solve it and obtain the optimal solution, then we sacrifice the optimality of the solution for speed and try to obtain a solution that is somewhat near to the optimal solution and can also be obtained in polynomial time and that's the idea behind approximation algorithms [4, 7, 27, 29]. Grid graphs, characterized by their structured layout where vertices correspond to points in a two-dimensional grid, present unique properties that can be exploited for optimization [28]. Their regularity allows for more straightforward analysis and algorithmic design compared to arbitrary graphs. Also, the structure of grid graphs, due to their regularity and boundedness, makes them particularly suitable for many algorithmic techniques. The problem of computing the independent dominating set (IDS) and dominating set (DS) retains its NP-hard complexity while also having additional combinatorial challenges. The independent dominating set is used in cases where resources must be placed without interference (e.g., sensor placement in networks). Understanding approximation algorithms in this context is crucial, especially in structured geometric graphs. Solving the minimum dominating set problem or computing the domination number of a graph is computationally complex for large graphs. Hence, the need to use approximate or heuristic approaches to these problems is important. In grid graphs, vertices are arranged in a rectangular grid, and edges connect adjacent vertices horizontally and vertically (see Figure 1).
Computer science and mathematics researchers have proposed three approaches to solve NP-hard problems:

- *Try to design algorithms for solving hard problems and accept their (worst-case) exponential complexity if they are efficient and fast enough for most of the problem instances (partition the set of all instances into two subsets: easy instances and hard instances).*

- *Try to design algorithms that provide solutions close to the optimal solutions (Approximation algorithms).*

- *Try to design algorithms with slowly increasing worst-case exponential time complexity (such as $O(1.2^n)$).*

In this paper, we present a 2-approximation algorithm for finding the independent dominating set in a grid graph with running time $O(n^2)$, indicating that it can find a solution that dominates the vertices of the graph using at most 2 times the optimal solution. The organization of the paper will be as follows:
In Section 2, we explain the necessary definitions. In Section 3, we show that the IDS problem is NP-hard in grid graphs. In Section 4, we present an approximation algorithm for IDS problem and analyze the running time and calculate the approximation factor of the presented algorithm, and Section 5 includes the conclusions.

2. Preliminaries and definitions

Let $G = (V, E)$ be a graph with vertex set $V = \{v_1, v_2, ..., v_n\}$. The open neighborhood of a vertex v is the set $\mathcal{N}(v) = \{u|(u, v) \in E\}$, the closed neighborhood of v is the set $\mathcal{N}[v] = \mathcal{N}(v) \cup \{v\}$. Similarly, the closed neighborhood of a set D is the set $\mathcal{N}[D] = \cup_{v \in D}\mathcal{N}[v]$. For an edge $e = (u, v)$, vertices u and v are the endpoint of edge e, and u is adjacent to v. We use $dist(u, v)$ to indicate the distance between u and v, which means the distance is the number of edges from the shortest path from u to v. For a vertex v have $\mathcal{N}_i(v) = \{u|dist(u, v) = i\}$ is defined as its ith level neighborhood, $\mathcal{N}^k(v) = \bigcup_{i=1}^{k} \mathcal{N}_i(v)$, and $\mathcal{N}_i[v] = \mathcal{N}_i(v) \cup \{v\}$. Particularly, $\mathcal{N}(v) = \mathcal{N}_1(v)$ and $\mathcal{N}[v] = \mathcal{N}_1[v]$. A vertex $v \in V$ is dominated by a candidate solution $CS \subseteq V$ if $v \in \mathcal{N}[u]$ and $u \in CS$, and is non-dominated otherwise. A set $D \subseteq V$ is a dominating set of G if every vertex in $V - D$ is adjacent to at least one vertex in D, or equivalently if $\mathcal{N}[D] = V$. The domination number $\gamma(G)$ of G equals the minimum cardinality of a dominating set D of G. We say that such a set D is a γ-set. For more details on domination theory, we refer the reader to [11–13]. For any $D \subseteq V$, the induced subgraph $G[D]$ is the graph whose vertex set is D and whose edge set consists of all edges in E that have both endpoints in D, that is, for $D \subseteq V$ a subgraph $G[D]$ is a subgraph of G obtained by deleting vertices $V \backslash D$ from G. In a graph G, a set D is called an independent dominating set if $\mathcal{N}[D] = V$, and no two vertices in D are adjacent.

An $m \times n$ grid graph $G_{n,m}$ has a vertex set $V = \{(i, j)|1 \leq i \leq n, 1 \leq j \leq m\}$ with (i, j) and adjacent to (k, l) if $i = k$, then $|j - l| = 1$ or if $j = l$, then $|i - k| = 1$ (see Figure 1). Several papers have been published on the domination numbers of classes of grid graphs, including [1–3, 10, 15, 26]. In [1, 10] the authors prove an exact formula for the domination number of $G_{n,m}$, for all $n, m \geq 1$. In [6, 14] the authors give methods for constructing γ-sets of $G_{n,m}$.

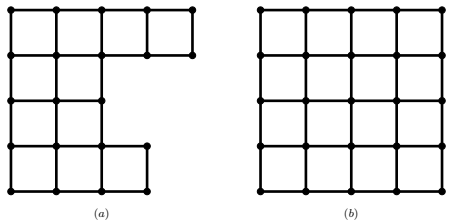


Figure 1. **(a) An incomplete grid graph G. (b) A complete grid graph $G_{5,5}$.**

3. NP-completeness results

In this section, we study the NP-completeness of an independent dominating set of grid graphs. Some of the NP-completeness results which are already known for the

Domination Decision problem, and Independent Domination Decision problem one can preferably see in [5, 8, 9, 16–18, 20, 25, 30].
Before stating the results regarding the NP-completeness of the independent dominating set of a network graph, the authors prefer to talk about the history of NP-completeness and the process of proving the NP-completeness of the independent dominating set of graphs.
The related work on the independent dominating set (IDS) problem is described as follows. Garey and Johnson [9] first showed that the IDS problem for general graphs is NP-complete. Then, Corneil and Perl [8] showed that the IDS problem for bipartite graphs is NP-complete. Later, Clark et al. [5] showed that the IDS problem for grid graphs is NP-complete. Zverovich and Zverovich [30] further showed that the IDS problem is NP-complete for at-most-cubic planar bipartite graphs of minimum girth of some fixed k, where the girth of a graph is the length of a shortest cycle contained in the graph. Moreover, Manlove [18] showed that the IDS problem for cubic planar graphs is NP-complete. Liu et al. [16] showed that the IDS problem for cubic bipartite graphs is NP-complete. Moreover, via reduction from the rectilinear planar monotone 3SAT problem, they obtained an alternative but more direct proof for the NP-completeness of the IDS problem on at-most-cubic grid graphs.
As mentioned above, the IDS problem is NP-complete in both the class of cubic planar graphs and the class of cubic bipartite graphs, and until 2020, the computational complexity of the problem in the intersection of these graph classes was unknown. Recently, Loverov and Orlovich [17] proved that the IDS problem is NP-complete in the class of cubic planar bipartite graphs.
The following two results are important.

Theorem 1. ([9, 20]) *The DOMINATION DECISION problem is NP-complete for bipartite graphs, chordal graphs and planar graphs. It also remains NP-complete for chordal bipartite graphs (a subclass of bipartite graphs) and split graphs (a subclass of chordal graphs).*

Theorem 2. ([5]) *The decision version of DS problem and IDS problem are NP-complete for grid graphs.*

The authors in [5] provide the above results for the decision version of the DS and IDS problem in grid graphs. In the decision version of these two problems we have, given a grid graph G and an integer k, is there a dominating set or independent dominating set of size k or less?
By Theorem 2, the decision version of the IDS problem is NP-complete on grid graphs. On the other hand, we know that an NP-complete problem is both in NP and NP-hard [9]. Hence, the decision version of the IDS problem is NP-hard on grid graphs. Finally, we know that if the decision version of a problem is NP-hard, then its non-decision (optimization) version is also NP-hard. Therefore, from Theorem 2 we can conclude that the optimization version of the IDS problem on grid graphs is an NP-hard problem.

4. The proposed approximation algorithm

In this section, we propose a 2-approximation algorithm to compute an independent dominating set of grid graphs. The algorithm employs a greedy approach, iteratively selecting vertices based on their potential to cover the maximum number of uncovered neighbors. Below, we provide the pseudocode for the proposed algorithm.

Algorithm 1: Compute_Scores(G, A,B,CS,W)

Input: A grid graph $G(V, E)$, array A, array B, set CS, set W .
Output: void (only update array A (compute score for all vertex of W)).
1: **for** each vertex $v_i \in W$ **do**
2: **if** $v_i \in CS$ **then**
3: $A[i]$ = -1;
4: **else**
5: **if** $|\mathcal{N}(v_i) \cap CS| = 0$ **then**
6: $A[i] = \Sigma(B[j] : v_j \in \mathcal{N}[v_i]$ *and* $|\mathcal{N}(v_j) \cap CS| = 0)$;
7: **else**
8: $A[i] = 0$;
9: **end of for**

Algorithm 2: Compute_CS(G)

Input: A grid graph $G(V, E)$.
Output: CS (An approximation solution for an independent dominating set in grid graphs).
1: Empty array A; Empty array B; Empty set CS; Empty set W;
2: $n = |V|$;
3: **for** $j = 1$ to n **do**
4: $B[j]= 1$;
5: W=V;
6: **while** ($\mathcal{N}[CS] \neq V$) **do**
7: Compute_Scores(G,A,B,CS,W); ▷ Update array A by Algorithm 1
8: Let v_m be vertex with the highest score in array A;
9: $CS = CS \cup \{v_m\}$;
10: $W = \mathcal{N}[v_m] \cup \mathcal{N}_2(v_m)$;
11: **for** $j = 1$ to n **do**
12: **if** $v_j \notin \mathcal{N}[CS]$ **then**
13: $++B[j]$; ▷ Update array B
14: **end of while**
15: **return** CS;

4.1. Analysis of Algorithm 2

Each vertex in graph G has two components (*score* and *cost*), and we also make a contract: $A[i] = score(v_i)$ and $B[j] = cost(v_j)$. Note that $|\mathcal{N}(v_i) \cap CS| = 0$ means that none of the neighbors of the vertex v_i are in the set CS.
Algorithm 1 is used to calculate the *score* of each vertex in the set W.
Algorithm 2 is implemented as follows:
First the *cost* of all graph vertices is set to one. In the *while* loop, first Algorithm 1 is called to calculate the *score*s of all vertices. Then the vertex with the highest *score* (v_m) is placed in the set CS. Next, the vertices in the $\mathcal{N}[v_m] \cup \mathcal{N}_2(v_m)$ are placed in the set W (this is because in the later iterations of executing the *while* loop, only the *scores* of vertices that are in the $\mathcal{N}[v_m] \cup \mathcal{N}_2(v_m)$ will change, because the *scores* of other vertices will remain unchanged and there is no need to recalculate them), and

finally, we increase the *cost* of vertices that are not yet dominated by the set CS by one unit (to increase their chances of being selected as the vertex candidate in the next iteration). The *while* loop repeats the same procedure until all vertices of the graph are dominated by vertices in the set CS.
Note that if at some iteration of the Algorithm 2 execution, several vertices, and not one vertex, have the largest *score* value among all vertices, we select the vertex with the smallest index in the array A. Note that the first iteration of the *while* loop is executed $|W| = n$ and on other iterations of the loop $|W| = |\mathcal{N}[v_m] \cup \mathcal{N}_2(v_m)| = |\{v_m\}| + |\mathcal{N}(v_m)| + |\mathcal{N}_2(v_m)| \leq (1 + 4 + 8) = 13$. See Figure 2.

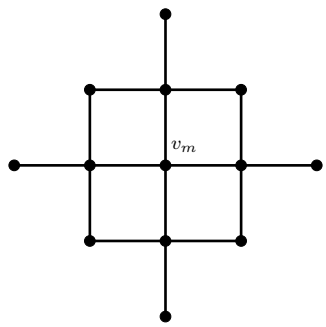


Figure 2. **At most vertices of $W = \mathcal{N}[v_m] \cup \mathcal{N}_2(v_m)$ ($|W| = 13$)**

We claim that the output of Algorithm 2 results in an independent dominating set, because each time the *while* loop is executed, using Algorithm 1, the array A is updated and the vertex with the highest *score* is selected as a candidate vertex to be included in set CS. And since the vertex with the highest *score* is calculated in the sixth line of Algorithm 1 and this line is executed when $|\mathcal{N}(v_i) \cap CS| = 0$, it means that the candidate vertex is not adjacent to any of the vertices of set CS, and this is the concept of independent dominating set.

In the following two examples, we present examples of running Algorithm 2 on an incomplete grid graph G_1 and a complete grid graph G_2.

Example 1. *In this example, we give an example of running Algorithm 2 on an incomplete grid graph G_1 as shown in Figure 3 (a). The steps of running the algorithm on this graph and its results for each iteration of the while loop are shown in Table 1. As can be seen in Figure 3, in this graph with 21 vertices ($|V| = 21$), the algorithm takes 8 iterations in the while loop, and the solution obtained by Algorithm 2 is $|CS| = 8$ (see Figure 3 (b)) when $CS = \{v_1, v_4, v_7, v_{10}, v_{14}, v_{16}, v_{19}, v_{21}\}$, and the optimal solution for this graph is $OPT = \{v_1, v_5, v_8, v_{11}, v_{17}, v_{19}\}$ and $|OPT| = 6$ as shown in Figure 3 (c).*

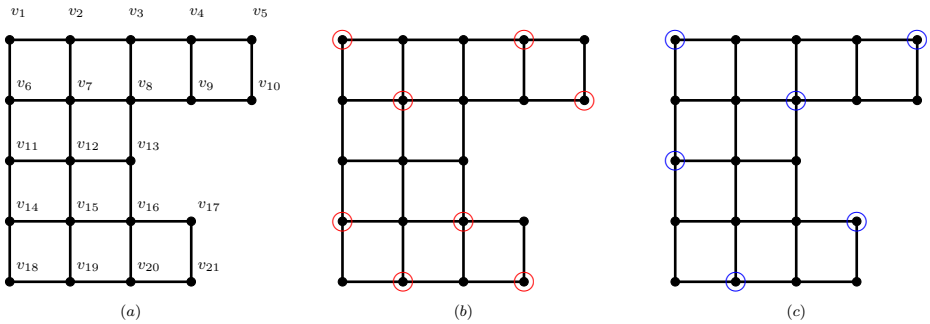


Figure 3. (a) An incomplete grid graph G_1. (b) An independent dominating set resulting from Algorithm 2. (c) An optimal independent dominating set.

Table 1. **Steps of running Algorithm 2 on an incomplete grid graph G_1.**

Iteration	CS	$B[j]$=**Cost**(v_j) $j = 1, 2, ..., 21$	$A[i]$=**Score**(v_i) $i = 1, 2, ..., 21$
1	$CS = CS \cup \{v_7\}$	[2, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2]	[3, 4, 4, 4, 3, 4, 5, 5, 4, 3, 4, 5, 4, 4, 5, 5, 3, 3, 4, 4, 3]
2	$CS = CS \cup \{v_{16}\}$	[3, 1, 3, 3, 3, 1, 1, 1, 3, 3, 3, 1, 2, 3, 2, 2, 2, 3, 3, 2, 3]	[2, 0, 4, 8, 6, 0, −1, 0, 6, 6, 4, 0, 4, 8, 8, 10, 6, 6, 8, 8, 6]
3	$CS = CS \cup \{v_4\}$	[4, 1, 3, 3, 3, 1, 1, 1, 3, 4, 4, 1, 2, 4, 2, 2, 2, 4, 4, 2, 4]	[3, 0, 6, 12, 9, 0, −1, 0, 9, 9, 6, 0, 0, 9, 0, −1, 0, 9, 6, 0, 3]
4	$CS = CS \cup \{v_{14}\}$	[5, 1, 3, 3, 3, 1, 1, 1, 3, 5, 4, 1, 2, 4, 2, 2, 2, 4, 5, 2, 5]	[4, 0, 0, −1, 0, 0, −1, 0, 0, 4, 8, 0, 0, 12, 0, −1, 0, 12, 8, 0, 4]
5	$CS = CS \cup \{v_1\}$	[5, 1, 3, 3, 3, 1, 1, 1, 3, 6, 4, 1, 2, 4, 2, 2, 2, 4, 6, 2, 6]	[5, 0, 0, −1, 0, 0, −1, 0, 0, 5, 0, 0, 0, −1, 0, −1, 0, 0, 5, 0, 5]
6	$CS = CS \cup \{v_{10}\}$	[5, 1, 3, 3, 3, 1, 1, 1, 3, 6, 4, 1, 2, 4, 2, 2, 2, 4, 7, 2, 7]	[−1, 0, 0, −1, 0, 0, −1, 0, 0, 6, 0, 0, 0, −1, 0, −1, 0, 0, 6, 0, 6]
7	$CS = CS \cup \{v_{19}\}$	[5, 1, 3, 3, 3, 1, 1, 1, 3, 6, 4, 1, 2, 4, 2, 2, 2, 4, 7, 2, 8]	[−1, 0, 0, −1, 0, 0, −1, 0, 0, −1, 0, 0, 0, −1, 0, −1, 0, 0, 7, 0, 7]
8	$CS = CS \cup \{v_{21}\}$	[5, 1, 3, 3, 3, 1, 1, 1, 3, 6, 4, 1, 2, 4, 2, 2, 2, 4, 7, 2, 8]	[−1, 0, 0, −1, 0, 0, −1, 0, 0, −1, 0, 0, 0, −1, 0, −1, 0, 0, −1, 0, 8]

Example 2. *In this example, we give an example of running Algorithm 2 on a complete grid graph G_2 as shown in Figure 4 (a). The steps of running the algorithm on this graph and its results for each iteration of the while loop are shown in Table 2. As can be seen in Figure 4, in this graph with 25 vertices ($|V| = 25$), the algorithm takes 8 iterations in while loop, and the solution obtained by Algorithm 2 is $|CS| = 8$ (see Figure 4 (b)) when $CS = \{v_1, v_4, v_7, v_{10}, v_{14}, v_{16}, v_{20}, v_{23}\}$, and the optimal solution for this graph is $OPT = \{v_3, v_6, v_{10}, v_{13}, v_{16}, v_{20}, v_{23}\}$ and $|OPT| = 7$ as shown in Figure 4 (c).*

4.2. Running time analysis of Algorithm 2

In this section, we analyze the running time of the proposed approximation algorithm. The running time of the algorithm can be broken down into several components based on its structure. First it is the initialization step to assigning the *cost* of all vertices to

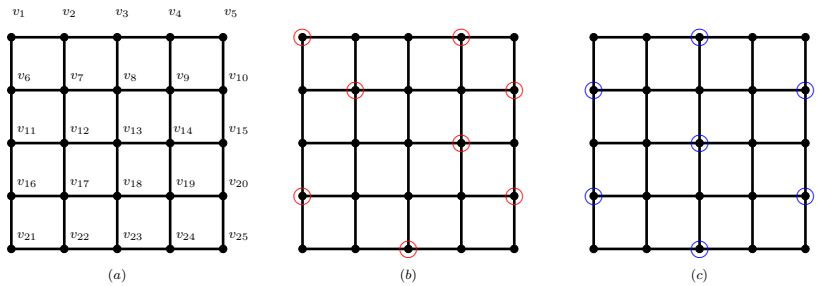


Figure 4. (a) A complete grid graph G_2. (b) An independent dominating set resulting from Algorithm 2. (c) An optimal independent dominating set.

Table 2. **Steps of running Algorithm 2 on a complete grid graph G_2.**

Iteration	CS	$B[j]$=**Cost**(v_j) $j = 1, 2, ..., 25$	$A[i]$=**Score**(v_i) $i = 1, 2, ..., 25$
1	$CS = CS \cup \{v_7\}$	[2, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]	[3, 4, 4, 4, 3, 4, 5, 5, 5, 4, 4, 5, 5, 5, 4, 4, 5, 5, 5, 4, 3, 4, 4, 4, 3]
2	$CS = CS \cup \{v_{14}\}$	[3, 1, 3, 3, 3, 1, 1, 1, 2, 3, 3, 1, 2, 2, 2, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3]	[2, 0, 4, 8, 6, 0, −1, 0, 8, 8, 4, 0, 6, 10, 8, 8, 8, 10, 10, 8, 6, 8, 8, 8, 6]
3	$CS = CS \cup \{v_{16}\}$	[4, 1, 4, 4, 4, 1, 1, 1, 2, 4, 3, 1, 2, 2, 2, 3, 3, 4, 2, 4, 3, 4, 4, 4, 4]	[3, 0, 6, 9, 9, 0, −1, 0, 0, 6, 6, 0, 0, −1, 0, 12, 12, 9, 0, 6, 9, 12, 12, 9, 9]
4	$CS = CS \cup \{v_{23}\}$	[5, 1, 5, 5, 5, 1, 1, 1, 2, 5, 3, 1, 2, 2, 2, 3, 3, 4, 2, 5, 3, 4, 4, 4, 5]	[4, 0, 8, 12, 12, 0, −1, 0, 0, 8, 0, 0, 0, −1, 0, −1, 0, 8, 0, 8, 0, 8, 16, 12, 12]
5	$CS = CS \cup \{v_4\}$	[6, 1, 5, 5, 5, 1, 1, 1, 2, 6, 3, 1, 2, 2, 2, 3, 3, 4, 2, 6, 3, 4, 4, 4, 6]	[5, 0, 10, 15, 15, 0, −1, 0, 0, 10, 0, 0, 0, −1, 0, −1, 0, 0, 0, 10, 0, 0, −1, 0, 10]
6	$CS = CS \cup \{v_{20}\}$	[7, 1, 5, 5, 5, 1, 1, 1, 2, 7, 3, 1, 2, 2, 2, 3, 3, 4, 2, 6, 3, 4, 4, 4, 6]	[6, 0, 0, −1, 0, 0, −1, 0, 0, 6, 0, 0, 0, −1, 0, −1, 0, 0, 0, 12, 0, 0, −1, 0, 12]
7	$CS = CS \cup \{v_1\}$	[7, 1, 5, 5, 5, 1, 1, 1, 2, 8, 3, 1, 2, 2, 2, 3, 3, 4, 2, 6, 3, 4, 4, 4, 6]	[7, 0, 0, −1, 0, 0, −1, 0, 0, 7, 0, 0, 0, −1, 0, −1, 0, 0, 0, −1, 0, 0, −1, 0, 0]
8	$CS = CS \cup \{v_{10}\}$	[7, 1, 5, 5, 5, 1, 1, 1, 2, 8, 3, 1, 2, 2, 2, 3, 3, 4, 2, 6, 3, 4, 4, 4, 6]	[−1, 0, 0, −1, 0, 0, −1, 0, 0, 8, 0, 0, 0, −1, 0, −1, 0, 0, 0, −1, 0, 0, −1, 0, 0]

the number one, which is of order $O(n)$. Now, we need to see how many iterations the *while* loop will have in the worst-case. Calling the function *Compute_scores* (updates the array A, which holds the *score* values of each vertex) takes $O(n)$ time. Since the greedy approach process generally leads to faster vertex dominance, finding the vertex with the highest *score* in the array A by max-heap tree takes $O(\log n)$. The addition of the selected vertex v_m to the CS set is of order $O(1)$. Updating the set

W with the $\mathcal{N}[v_m] \cup \mathcal{N}(v_m)$ vertices is of order $O(1)$ (because the maximum number of vertices of the W set is 13 (see Figure 2)). Updating the array B for vertices not currently dominated by CS set has $O(n)$ time. Now, because in the worst-case, at least two vertices and at most five vertices are dominated with each iteration of the *while* loop (including the candidate vertex v_m), Then, to dominate all vertices of the graph (depending on the given graph instance), the *while* loop is iterated at least $\lfloor n/4 \rfloor$ and at most $\lceil n/2 \rceil$ times due to the property of grid graphs that each vertex has at most 4 neighbors, which means that the *while* loop takes $O(n)$ time. Therefore, the time complexity of the proposed approximation algorithm is:
$O(n) + O(n).[O(n) + O(\log n) + O(n)] = O(n) + O(n) \times O(n) = O(n) + O(n^2) = O(n^2)$.

4.3. Running time analysis of Algorithm 2

In this section, we analyze the running time of the proposed approximation algorithm. The running time of the algorithm can be broken down into several components based on its structure. First it is the initialization step to assigning the *cost* of all vertices to the number one, which is of order $O(n)$. Now, we need to see how many iterations the *while* loop will have in the worst-case. Calling the function *Compute_scores* (updates the array A, which holds the *score* values of each vertex) takes $O(n)$ time. Since the greedy approach process generally leads to faster vertex dominance, finding the vertex with the highest *score* in the array A by max-heap tree takes $O(\log n)$. The addition of the selected vertex v_m to the CS set is of order $O(1)$. Updating the set W with the $\mathcal{N}[v_m] \cup \mathcal{N}(v_m)$ vertices is of order $O(1)$ (because the maximum number of vertices of the W set is 13 (see Figure 2)). Updating the array B for vertices not currently dominated by CS set has $O(n)$ time. Now, because in the worst-case, at least two vertices and at most five vertices are dominated with each iteration of the *while* loop (including the candidate vertex v_m), Then, to dominate all vertices of the graph (depending on the given graph instance), the *while* loop is iterated at least $\lfloor n/4 \rfloor$ and at most $\lceil n/2 \rceil$ times due to the property of grid graphs that each vertex has at most 4 neighbors, which means that the *while* loop takes $O(n)$ time. Therefore, the time complexity of the proposed approximation algorithm is:
$O(n) + O(n).[O(n) + O(\log n) + O(n)] = O(n) + O(n) \times O(n) = O(n) + O(n^2) = O(n^2)$.

Theorem 3. *Algorithm 2 can compute an approximate solution to the problem of computing the independent dominating set on a grid graph in $O(n^2)$ time.*

Proof. The algorithm iteratively selects the vertex with the maximum *score* until all vertices are dominated by the set CS. Assigning the number one to the elements of array B is of order $O(n)$. Extracting the maximum *score* from the max-heap tree requires $O(1)$ due to the properties of the max-heap tree. But, removing the maximum element and re-establishing the heap property (by max-heapifying) takes $O(\log n)$ time in the worst case in each iteration of the *while* loop. After selecting

the vertex v_m , by using the W set, The time complexity for adjacent vertices in the 2×2 range by the layered range tree is $O(k)$, where k is the number of vertices in the 2×2 range around the vertex v_m with center point v_m and since k is constant (maximum 13 in a grid), we simplify this to $O(1)$. Time complexity for update array B is of order $O(n)$. Thus, the total time for each iteration of the *while* loop simplifies to:$O(n) + O(\log n) + O(n) = O(n)$. Since the *while* loop runs for at most $\lceil n/2 \rceil$ iterations (in the worst-case), which is of the order $O(n)$, the total running time across all iterations is $O(n) + O(n).[O(n) + O(\log n) + O(n)] = O(n^2)$. Therefore, we have demonstrated that the running time of Algorithm 2 is $O(n^2)$. □

4.4. Approximation factor of the Algorithm 2

In this section, we examine the approximation factor of the proposed algorithm and show that this algorithm has an approximation factor of 2. First, we present the following lemma about grid graphs.

Lemma 1. *By choosing at most $\lceil n/2 \rceil$ vertices in a grid graph, one can create an independent dominating set where $n = |V|$.*

Proof. Let D be the independent dominating set that we want to create for a grid graph. Consider the following process: Starting from the top level of the grid graph, mark a vertex and add it as a member of the set D. Then traverse the entire graph level by level and for each edge of the graph, if none of its vertices are marked, select and mark one of the vertices. Repeat this process for all the edges of the graph. In this case, it is clear that the set D contains $\lceil n/2 \rceil$ vertices and D is an independent dominating set. Since for each edge, one of its vertices is selected, and therefore all the vertices of the graph are dominated. On the other hand, adding any other vertex of $V \backslash D$ in the grid graph to D adds an extra vertex to D, because it does not dominate a new vertex of the graph. Therefore, by selecting at most $\lceil n/2 \rceil$ vertices in a grid graph, we will have an independent dominating set and the proof is complete. □

Theorem 4. *The proposed approximation algorithm provides a 2-approximation solution to the problem of computing an independent dominating set in a grid graph.*

Proof. Suppose CS is the solution set obtained from Algorithm 2 and OPT is the optimal solution for a given grid graph. Given that in Algorithm 2, the vertex with the maximum **score** is selected in each iteration, and given the CS calculation process in Algorithm 2, which prevents the selection of two adjacent vertices in the solution set, the number of vertices selected in CS cannot be greater than the number of vertices in the set D in Lemma 1, and we have: $|CS| \leq |D| = \lceil n/2 \rceil$. If $|OPT|$ is the number of vertices of an optimal independent dominating set in a grid graph, then for each grid graph we will have: $\lfloor n/4 \rfloor \leq |OPT|$ (due to the property of grid graphs that each vertex has at most 4 neighbors). From the above relations, we have:$\lfloor n/4 \rfloor \leq |OPT| \leq |CS| \leq 2 \cdot (\lfloor n/4 \rfloor)$. Therefore, $|OPT| \leq |CS| \leq 2 \cdot |OPT|$,

and the number of vertices in the solution CS of Algorithm 2 is at most 2 times the optimal solution OPT, and therefore, the proposed approximation algorithm is a 2-approximation and this completes the proof. □

5. Conclusion

Given that the dominating set and its types are known as an NP-hard problem. In this paper, we study the problem of computing the independent dominating set in grid graphs and propose a 2-approximation algorithm for an independent dominating set problem with the running time $O(n^2)$, This polynomial time complexity makes the algorithm efficient and suitable for practical use on reasonably sized grid graphs, ensuring it can compute an approximate independent dominating set within a feasible solution.

Conflict of Interest: The authors declare that they have no conflict of interest.

Data Availability: Data sharing is not applicable to this article as no datasets were generated or analyzed during the current study.

References

[1] S. Alanko, S. Crevals, A. Isopoussu, P. Ostergard, and V. Pettersson, *Computing the domination number of grid graphs*, Electron. J. Combin. **18** (2011), no. 1, #P141.
https://doi.org/10.37236/628.

[2] T.Y. Chang and W.E. Clark, *The domination numbers of the* $5 \times n$ *and* $6 \times n$ *grid graphs*, J. Graph Theory **17** (1993), no. 1, 81–107.
https://doi.org/10.1002/JGT.3190170110.

[3] T.Y. Chang, W.E. Clark, and E.O. Hare, *Dominations of complete grid graphs, I*, Ars Combin. **38** (1994), 97–112.

[4] M. Chlebik and J. Chlebikova, *Approximation hardness of dominating set problems in bounded degree graphs*, Inform. and Comput. **206** (2008), no. 11, 1264–1275.
https://doi.org/10.1016/j.ic.2008.07.003.

[5] B.N. Clark, C.J. Colbourn, and D.S. Johnson, *Unit disk graphs*, Discrete Math. **86** (1990), no. 1-3, 165–177.
https://doi.org/10.1016/0012-365X(90)90358-O.

[6] E.J. Cockayne, E.O. Hare, S.T. Hedetniemi, and T.V. Wymer, *Bounds for the domination number of grid graphs*, Congr. Numer. **47** (1985), 217–228.

[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, Prentice Hall, India, 2001.

[8] D.G. Corneil and Y. Perl, *Clustering and domination in perfect graphs*, Discrete Appl. Math. **9** (1984), no. 1, 27–39. https://doi.org/10.1016/0166-218X(84)90088-X.

[9] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, New York, 1979.

[10] D. Goncalves, A. Pinlou, M. Rao, and S. Thomasse, *The domination number of grids*, SIAM J. Discrete Math. **25** (2011), no. 3, 1443–1453. https://doi.org/10.1137/11082574.

[11] T.W. Haynes, S.T. Hedetniemi, and P.J. Slater, *Domination in Graphs, Advanced Topics*, Marcel Dekker, 1998.

[12] ———, *Fundamentals of Domination in Graphs*, Marcel Dekker, Inc., 1998.

[13] T.W. Haynes, M.A. Henning, and S.T. Hedetniemi, *Domination in Graphs: Core Concepts*, Springer, 2023.

[14] K.R. Hutson, S.T. Hedetniemi, and R.J. Forrester, *Constructing γ-sets of grids*, J. Combin. Math. Combin. Comput. **95** (2015), 3–26.

[15] M.S. Jacobson and L.F. Kinch, *On the domination number of products of a graph, I*, Ars Combin. **18** (1983), 33–44.

[16] C.H. Liu, S.H. Poon, and J.Y. Lin, *Independent dominating set problem revisited*, Theoret. Comput. Sci. **562** (2015), 1–22. https://doi.org/10.1016/j.tcs.2014.09.001.

[17] Y.A. Loverov and Y.L. Orlovich, *NP-completeness of the independent dominating set problem in the class of cubic planar bipartite graphs*, J. Appl. Ind. Math. **14** (2020), no. 2, 353–368. https://doi.org/10.1134/S1990478920020131.

[18] D.F. Manlove, *On the algorithmic complexity of twelve covering and independence parameters of graphs*, Discrete Appl. Math. **91** (1999), no. 1-3, 155–175. https://doi.org/10.1016/S0166-218X(98)00147-4.

[19] M.A. Meybodi and A. Poureidi, *On $[1,2]$-domination in interval and circle graphs*, Discrete Math. Theoret. Comput. Sci. **26** (2024), #6.

[20] H. Müller and A. Brandstädt, *The NP-completeness of steiner tree and dominating set for chordal bipartite graphs*, Theoret. Comput. Sci. **53** (1987), no. 2-3, 257–265. https://doi.org/10.1016/0304-3975(87)90067-3.

[21] A. Poureidi, *Efficient algorithms for independent Roman domination on some classes of graphs*, Commun. Comb. Optim. **8** (2021), no. 1, 127–140. https://doi.org/10.22049/cco.2021.27305.1228.

[22] ———, *Algorithm and hardness results in double Roman domination of graphs*, Theoret. Comput. Sci. **911** (2022), 70–79. https://doi.org/10.1016/j.tcs.2022.02.006.

[23] ———, *Double Roman domination in graphs: Algorithmic complexity*, Commun. Comb. Optim. **8** (2023), no. 3, 491–503. https://doi.org/10.22049/CCO.2022.27661.1309.

[24] A. Poureidi and J. Fathali, *Algorithmic complexity of triple Roman dominating functions on graphs*, Commun. Comb. Optim. **9** (2024), no. 2, 217–232.

https://doi.org/10.22049/cco.2023.27884.1385.

[25] Y. Song, T. Liu, and K. Xu, *Independent domination on tree convex bipartite graphs*, Proceedings 6th FAW-AAIM, 2012, pp. 129–138.

[26] A. Spalding, *A min plus algebra scheme for computing domination numbers*, Ph.D. thesis, Department of Mathematics, University of Colorado, 2001.

[27] J.A. Telle, *Complexity of domination-type problems in graphs*, Nord. J. Comput. **1** (1994), no. 1, 157–171.
https://doi.org/10.5555/640186.640195.

[28] D.B. West, *Introduction to Graph Theory*, second ed., Prentice Hall, USA, 2001.

[29] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre, *A simple model to generate hard satisfiable instances*, International Joint Conference on Artificial Intelligence, 2005, pp. 337–342.
https://doi.org/10.5555/1642293.1642347.

[30] I.E. Zverovich and V.E. Zverovich, *An induced subgraph characterization of domination perfect graphs*, J. Graph Theory **20** (1995), no. 3, 375–395.
https://doi.org/10.1002/jgt.3190200313.