Exploring Dominating Functions and Their Complexity in Subclasses of Weighted Chordal Graphs and Bipartite Graphs

Chuan-Min Lee *

Posted Date: 27 November 2024

doi: 10.20944/preprints202411.2117.v1

Keywords: domination; total domination; {k}-domination; k-tuple domination; strongly chordal graph; chordal bipapartite graph; proper interval graph; convex bipartite graph; totally balanced matrix; totally unimodular matrix

Article

Exploring Dominating Functions and Their Complexity in Subclasses of Weighted Chordal Graphs and Bipartite Graphs

Chuan-MIn Lee

Department of Applied Artificial Intelligence, Ming Chuan University, 5 De Ming Road, Guishan District, Taoyuan City 333, Taiwan; joneslee@mail.mcu.edu.tw; Tel.: +886-3-350-7001 (ext. 3438)

Abstract: Domination problems are fundamental problems in graph theory with diverse applications in optimization, network design, and computational complexity. This paper investigates $\{k\}$-domination, k-tuple domination, and their total domination variants in weighted strongly chordal graphs and chordal bipartite graphs—two well-studied subclasses of chordal graphs and bipartite graphs. We extend existing theoretical models to explore the less-explored domain of vertex-weighted graphs and establish efficient algorithms for these domination problems. Specifically, the $\{k\}$-domination problem in weighted strongly chordal graphs and the total $\{k\}$-domination problem in weighted chordal bipartite graphs are shown to be solvable in $\mathcal{O}(n+m)$ time. For weighted proper interval graphs and convex bipartite graphs, we solve the k-tuple domination and total k-tuple domination problems in $\mathcal{O}(n^{2.371552}\log^2(n)\log(n/\delta))$, where δ is the desired accuracy. Furthermore, for weighted unit interval graphs, the k-tuple domination problem achieves a significant complexity improvement, reduced from $\mathcal{O}(n^{k+2})$ to $\mathcal{O}(n^{2.371552}\log^2(n)\log(n/\delta))$. These results are achieved through a combination of linear and integer programming techniques, complemented by totally balanced matrices, totally unimodular matrices, and graph-specific matrix representations such as neighborhood and closed neighborhood matrices.

Keywords: domination; total domination; $\{k\}$-domination; k-tuple domination; strongly chordal graph; chordal bipapartite graph; proper interval graph; convex bipartite graph; totally balanced matrix; totally unimodular matrix

1. Introduction

Domination is a fundamental concept in graph theory and widely regarded as a critical graph optimization problem. Over the years, it has attracted significant attention due to its diverse applications in network design, resource optimization, social network analysis, and computational complexity. Its importance was first notably recognized in 1990, when a special issue of *Discrete Mathematics* was dedicated entirely to this topic [1]. By 1998, more than 1,200 papers had been published on domination and its variants, which led to the release of two seminal books on the subject [2,3]. Since then, the field has experienced rapid growth, with over 5,000 papers exploring various types of domination in graphs. Three additional books [4–6] have been published in recent years to capture the latest advancements in domination theory. Among these variants, *total domination* has gained particular prominence and has become a well-researched problem, as outlined in [7].

1.1. Current Research Directions

An active area of current research focuses on $\{k\}$-*domination* and *total* $\{k\}$-*domination*. Domination and total domination are special cases when $k = 1$. Extensive studies on these problems have revealed insights into computational complexity and solution techniques [8–11,15,20–33]. For instance, $\{k\}$-domination is polynomial-time solvable for strongly chordal graphs, but remains NP-complete for chordal bipartite graphs [8]. Similarly, total $\{k\}$-domination is solvable in polynomial time for chordal bipartite graphs [9] but NP-complete for bipartite planar graphs [10]. From an approximation perspective, both problems can be approximated within a factor of $\ln n$ in polynomial time, where n is the number of vertices in the graph [11].

 doi:10.20944/preprints202411.2117.v1

Beyond $\{k\}$-domination, the *k-tuple domination* and *total k-tuple domination* problems have also garnered attention. While polynomial-time solutions exist for specific graph families, such as strongly chordal graphs, interval graphs, and web graphs [12–14], the problems remain NP-complete for others, such as chordal and planar graphs [12,15]. The k-tuple total domination problem can be solved in $\mathcal{O}(n^{3k})$ time for weighed proper interval graphs [16]. Furthermore, for $k < 4$, the running time of $\mathcal{O}(n^{3k})$ can be improved to be $\mathcal{O}(n^{2k+4})$. For weighted unit interval graphs, the k-tuple and total k-tuple domination problems can be solved in $\mathcal{O}(n^{k+2})$ and $\mathcal{O}(n^{k+3})$ time [17], respectively. The dichotomy between tractable and intractable cases has also been a focus of study in [36–43].

1.2. Research Gap and Motivation

Despite significant advancements, most studies on (total) $\{k\}$-domination and (total) k-tuple domination have concentrated on *unweighted* graphs. Comparatively, much less attention has been given to these problems in *weighted* graphs, despite their widespread use in real-world systems to model costs, capacities, or priorities. Exploring these domination problems in weighted graphs offers a significant opportunity for further research and development.

To bridge this gap, we focus on the $\{k\}$-domination and total $\{k\}$-tuple domination problems for weighted strongly chordal graphs and chordal bipartite graphs. Additionally, we investigate k-tuple domination and total k-tuple domination for weighted proper interval graphs and weighted convex bipartite graphs.

1.3. Scope of Study

Strongly chordal graphs and chordal bipartite graphs are well-studied classes that include several important subclasses, such as trees, proper interval graphs, convex bipartite graphs, and bipartite permutation graphs [18]. These classes are characterized by specific vertex ordering constraints, which facilitate various computational operations. Employing these properties, we aim to develop efficient algorithms for solving domination problems on weighted graphs.

Most existing research on domination in weighted graphs assigns weights to vertices, with each vertex's weight representing attributes like importance, cost, or influence that directly impact domination parameters. Following this convention, this paper focuses exclusively on vertex-weighted graphs, and the term "weighted graph" refers specifically to this context.

1.4. Contributions

This paper presents the following contributions:

1. **$\{k\}$-Domination for Weighted Strongly Chordal Graphs**: We demonstrate that the $\{k\}$-domination problem in weighted strongly chordal graphs can be solved in $\mathcal{O}(n+m)$ time, where n and m denote the number of vertices and edges, respectively. This result is achieved by refining Hoffman et al.'s method [19] and leveraging properties of adjacency lists and totally balanced matrices.
2. **Total $\{k\}$-Domination for Weighted Chordal Bipartite Graphs**: Building on structural similarities between chordal bipartite graphs and strongly chordal graphs, we extend the refined framework to solve the total $\{k\}$-domination problem in $\mathcal{O}(n+m)$ time for weighted chordal bipartite graphs.
3. ***k*-Tuple Domination for Weighted Proper Interval Graphs**: Using linear and integer linear programming techniques, we solve the k-tuple domination problem for weighted proper interval graphs in $\mathcal{O}\left(n^{2.371552}\log^2(n)\log(n/\delta)\right)$ time, where δ is the desired accuracy. Since a graph is a proper interval graph if and only if it is a unit interval graph [18], we reduce the complexity from $\mathcal{O}(n^{k+2})$ [17] to $\mathcal{O}\left(n^{2.371552}\log^2(n)\log\left(\frac{n}{\delta}\right)\right)$.
4. **Total *k*-Tuple Domination for Weighted Convex Bipartite Graphs**: We achieve the same time complexity bound, $\mathcal{O}\left(n^{2.371552}\log^2(n)\log\left(\frac{n}{\delta}\right)\right)$, for solving the total k-tuple domination problem in weighted convex bipartite graphs.

 doi:10.20944/preprints202411.2117.v1

This study is purely theoretical and includes formal proofs of correctness and efficiency, enriching the understanding of domination problems in these graph classes.

1.5. Organization of the Paper

The remainder of the paper is organized as follows:

- **Section 2: Preliminaries**

  This section provides a foundation of essential concepts and definitions that underpin our research. We introduce and explore fundamental properties of strongly chordal graphs, chordal bipartite graphs, proper interval graphs, and convex bipartite graphs. To set the stage for our linear programming approach, we formally define (total) $\{k\}$-domination and (total) k-tuple domination. Additionally, we review matrix-theoretic tools, specifically *adjacency matrices*, *totally balanced matrices*, *greedy matrices*, and *totally unimodular matrices*, which play a critical role in our modeling framework.
- **Section 3: $\{k\}$-Domination in Weighted Strongly Chordal Graphs**

  This section formulates the $\{k\}$-domination problem an integer linear program in matrix form. We demonstrate how to solve the $\{k\}$-domination problem for weighted strongly chordal graphs in $\mathcal{O}(n+m)$ time by refining Hoffman et al.'s method [19] using concepts from totally balanced matrices, greedy matrices, and adjacency lists.
- **Section 4: Total $\{k\}$-Domination in Weighted Chordal Bipartite Graphs**

  This section describes how to adapt the matrix-based framework to solve the total $\{k\}$-domination problem in weighted chordal bipartite graphs. By using the structural similarities between chordal bipartite and strongly chordal graphs, we demonstrate the versatility of our approach and provide a comprehensive analysis of its applicability to both graph types.
- **Section 5: k-Tuple Domination and Total k-Tuple Domination**

  This section focuses on using linear and integer linear programming for *totally unimodular matrices* to solve the k-tuple domination problem for weighted proper interval graphs and the total k-tuple domination problem for weighted convex bipartite graphs in $\mathcal{O}\left(n^{2.371552}\log^2(n)\log(n/\delta)\right)$ time.
- **Section 6: Conclusions**

  The paper concludes by summarizing our findings, discussing the theoretical implications of our results, and proposing directions for future research that extends these domination techniques to other classes of weighted graphs.

2. Preliminaries

This paper integrates critical insights from multiple fields, including graph classes, linear algebra, graph theory, algorithms, and linear programming, each contributing indispensable tools and perspectives that enhance both the formulation and depth of our analysis. This section presents the fundamental concepts most pertinent to understanding the study's objectives and methodology. For definitions, notations, additional concepts, or more comprehensive details not covered in this section, standard textbooks and monographs—such as *Introduction to Algorithms* [44], *Theory of Linear and Integer Programming* [45], *Graph Theory* [46], *Introduction to Linear Algebra* [47], and *Graph Class: A Survey* [18]—are recommended as supplementary resources.

2.1. Graphs and Their Representations

Each graph $G = (V, E)$ in this paper is finite, undirected, and contains no multiple edges or self-loops, where V is the vertex set and E is the edge set of G. If the vertex and edge sets are not explicitly specified, they are denoted by $V(G)$ and $E(G)$, respectively.

Two vertices u and v in a graph G are *adjacent* if they are connected with an edge, i.e., $(u, v) \in E(G)$, and are also called *neighbors*. The *degree* of a vertex v in G, denoted by $deg_G(v)$, is the number of neighbors of v. For any vertex $v \in V(G)$, the *neighborhood* of v in G, denoted by $N_G(v)$, is the set of

 doi:10.20944/preprints202411.2117.v1

neighbors of v, so that $|N_G(v)| = deg_G(v)$. A vertex u is a *closed neighbor* of v if either $u \in N_G(v)$ or $u = v$. The *closed neighborhood* of v, denoted by $N_G[v]$, is the set of closed neighbors of v.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. If $V' \subseteq V$ and $E' \subseteq E$, then G' is a *subgraph* of G. If G' is a subraph of G, and G' contains all the edge $(u, v) \in E$ with $u, v \in V'$, then G' is an *induced subgraph*; we say that G' is *induced* by V' and written as $G[V']$.

Graphs in this paper are represented using *adjacency lists*. The *adjacency-list representation* of a graph $G = (V, E)$ consists of an array of $|V|$ lists. Each vertex $v \in V$ is associated with a list of all neighbors of v. Figure 1 provides an example of the adjacency-list representation for a graph G with 4 vertices and 4 edges.

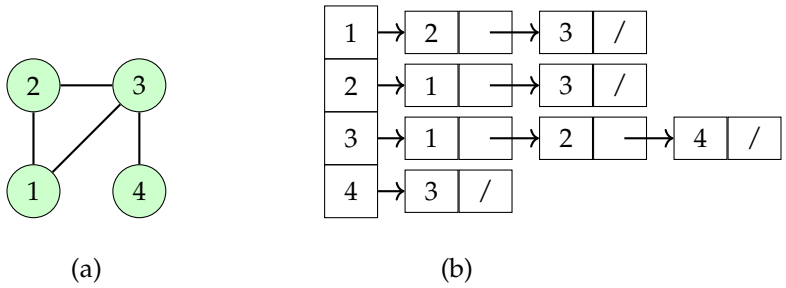


Figure 1. A representation of a graph G. (a) The graph G with 4 vertices and 4 edges. (b) The adjacency-list representation of G.

This representation is efficient in terms of memory usage. The amount of memory required to store a graph in this representation is $\mathcal{O}(n + m)$, where n and m represent the number of vertices and edges, respectively. Adjacency lists support efficient operations such as

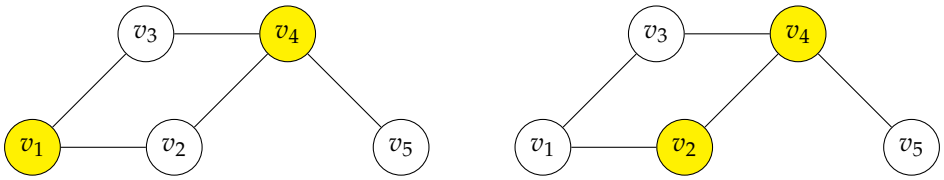
- *Neighbor Access*: Accessing all (closed) neighbors of a vertex v takes $\mathcal{O}(deg(v) + 1)$ time, as they are directly stored in a list. With each vertex maintaining its own neighbor list, this structure enables quick access to both neighbors and closed neighbors.
- *Traversal*: Traversing all vertices and edges in the graph takes $\mathcal{O}(n + m)$ time. This efficiency is particularly advantageous in algorithms like Depth-First Search (DFS) and Breadth-First Search (BFS).

2.2. $\{k\}$-Domination and Total $\{k\}$-Domination in Weighted Graphs

A dominating set D of a graph $G = (V, E)$ is a subset of V such that $|N_G[v] \cap D| \geq 1$ for every $v \in V$, while a total dominating set D is a subset of V such that $|N_G(v) \cap D| \geq 1$ for every $v \in V$. The domination number of G, denoted by $\gamma(G)$, is the minimum cardinality of a dominating set of G. The total domination number of G, denoted by $\gamma_t(G)$, is the minimum cardinality of a total dominating set of G. The domination problem is to find a dominating set of G with the minimum cardinality, and the total domination problem is to find a total dominating set of G with the minimum cardinality.

Figure 2 illustrates a dominating set $D_1 = \{v_1, v_4\}$ and a total dominating set $D_2 = \{v_2, v_4\}$ for the same graph. Let G denote the graph in this figure. While D_2 also qualifies as a dominating set of G, D_1 does not qualify as a total dominating set, since $N_G(v_i) \cap D_1$ is empty for $i \in \{1, 4\}$.

 doi:10.20944/preprints202411.2117.v1



(a) (b)

Figure 2. Two sets $D_1 = \{v_1, v_4\}$ and $D_2 = \{v_2, v_4\}$. (a) D_1 is a dominating set. (b) D_2 is a total dominating set.

Domination and total domination can be expressed in terms of functions. Let $f : V \to \{0, 1\}$ be a labeling function of a graph $G = (V, E)$. The function f is a dominating (respectively, total dominating) function of G if $\sum_{v \in N_G[v]} f(v) \geq 1$ (respectively, $\sum_{v \in N_G(v)} f(v) \geq 1$) for every $v \in V$. The labeling weight of f is defined as $\sum_{v \in V} f(v)$.

Table 1 presents a dominating function f and a total dominating function g for the graph shown in Figure 2. The function f is associated with the set D_1 and g with the set D_2. Both functions have a labeling weight of 2, equal to $|D_1|$ and $|D_2|$, respectively.

Table 1. Values of dominating functions f and g, and membership in sets D_1 and D_2 for the graph in Figure 2.

Vertex v	$f(v)$	$g(v)$	In D_1?	In D_2?
v_1	1	0	Yes	No
v_2	0	1	No	Yes
v_3	0	0	No	No
v_4	1	1	Yes	Yes
v_5	0	0	No	No

Clearly, the domination number $\gamma(G)$ is the minimum labeling weight of a dominating function, i.e.,

$$\gamma(G) = \min\{\sum_{v \in V} f(v) \mid f \text{ is a dominating function of } G\}.$$

Similarly, the total domination number $\gamma_t(G)$ is the minimum labeling weight of a total dominating function, i.e.,

$$\gamma_t(G) = \min\left\{\sum_{v \in V} f(v) \mid f \text{ is a total dominating function of } G\right\}.$$

Definition 1. *Let k be a fixed positive integer, and let $f : V \to \{0, 1, \ldots, k\}$ be a labeling function of a graph $G = (V, E)$. The function f is a $\{k\}$-dominating (respectively, total $\{k\}$-dominating) function of G if $\sum_{v \in N_G[v]} f(v) \geq k$ (respectively, $\sum_{v \in N_G(v)} f(v) \geq k$) for every $v \in V$. The labeling weight of f is defined as $\sum_{v \in V} f(v)$. The $\{k\}$-domination problem is to find a $\{k\}$-dominating function of G with the minimum labeling weight, while the total $\{k\}$-domination problem is to find a total $\{k\}$-dominating function of G with the minimum labeling weight.*

By Definition 1, domination and total domination correspond to $\{1\}$-domination and total $\{1\}$-domination, respectively.

Let G be the graph shown in Figure 2. Table 2 provides the values of a function f for each vertex in G and checks whether the $\{2\}$-domination condition is satisfied. This verification confirms that f is a valid $\{2\}$-dominating function for G.

 doi:10.20944/preprints202411.2117.v1

Table 2. Values of a $\{2\}$-dominating function f for each vertex of the graph G in Figure 2

Vertex v	$f(v)$	$N_G[v]$	$\sum_{u\in N_G[v]} f(u)$
v_1	0	$\{v_1, v_2, v_3\}$	$f(v_1)+f(v_2)+f(v_3)=2$
v_2	1	$\{v_1, v_2, v_4\}$	$f(v_1)+f(v_2)+f(v_4)=3$
v_3	1	$\{v_1, v_3, v_4\}$	$f(v_1)+f(v_3)+f(v_4)=3$
v_4	2	$\{v_2, v_3, v_4, v_5\}$	$f(v_2)+f(v_3)+f(v_4)+f(v_5)=4$
v_5	0	$\{v_4, v_5\}$	$f(v_4)+f(v_5)=2$

We now introduce the concepts of $\{k\}$-domination and total $\{k\}$-domination for weighted graphs. As mentioned earlier in the introduction, this paper focuses exclusively on vertex-weighted graphs.

Definition 2. *Let $w : V \to \mathbb{R}$ be a function that assigns a weight $w(v)$ to each vertex v of a graph $G = (V, E)$. We refer to w as a vertex-weight function, and (G, w) as a weighted graph.*

Definition 3. *Let k be a fixed positive integer. The labeling weight of a $\{k\}$-dominating function or a total $\{k\}$-dominating function f of a weighted graph (G, w) is defined as $\sum_{v\in V(G)} f(v) \cdot w(v)$. The $\{k\}$-domination problem for a weighted graph (G, w) is to find a $\{k\}$-dominating function of G with the minimum labeling weight, while the total $\{k\}$-domination problem is to find a total $\{k\}$-dominating function with the minimum labeling weight.*

An unweighted graph H can be treated as a specific case of a weighted graph, where every vertex $v \in V(H)$ has a weight $w(v) = 1$. The primary difference in defining the $\{k\}$-domination problem for unweighted and weighted graphs lies in the calculation of the *labeling weight*.

- **In unweighted graphs**, the labeling weight of a $\{k\}$-dominating function is simply the sum of the function values assigned to each vertex, i.e., $\sum_{v\in V} f(v)$, where each vertex has an implicit weight of 1.
- **In weighted graphs**, each vertex is assigned a specific weight through a *vertex-weight function* $w(v)$. The labeling weight is then calculated as $\sum_{v\in V} f(v) \cdot w(v)$, meaning that the contribution of each vertex to the total weight depends on both the value of the labeling function $f(v)$ and the vertex's weight $w(v)$.

Table 3 presents the values of a $\{2\}$-dominating function f and vertex weights w for the graph $G = (V, E)$ shown in Figure 2. The table indicates that the labeling weight of f for the unweighted graph G is 4, based on $\sum_{v\in V} f(v)$, and the labeling weight for the weighted graph (G, w) is 12, calculated as $\sum_{v\in V} f(v) \cdot w(v)$.

Table 3. Labeling weights of the $\{2\}$-dominating function f for the unweighted and weighted graphs.

Vertex v	**Weight** $w(v)$	$f(v)$	$f(v) \cdot w(v)$
v_1	2	0	0
v_2	3	1	3
v_3	1	1	1
v_4	4	2	8
v_5	2	0	0
Labeling Weight		4	12

This distinction also applies to the definitions of total $\{k\}$-domination on unweighted and weighted graphs. Therefore, the introduction of vertex weights in weighted graphs alters how the overall labeling weight is determined in both the $\{k\}$-domination and total $\{k\}$-domination problems.

 doi:10.20944/preprints202411.2117.v1

2.3. k-Tuple Domination and Total k-Tuple Domination in Weighted Graphs

Let k be a fixed positive integer, and let $f : V \to \{0,1\}$ be a labeling function for a graph $G = (V, E)$. The function f is a k-tuple dominating function of G if $\sum_{v \in N_G[v]} f(v) \geq k$ for every $v \in V$. Similarly, f is a total k-tuple dominating function if $\sum_{v \in N_G(v)} f(v) \geq k$ for every $v \in V$. The labeling weight of f is defined as $\sum_{v \in V} f(v)$.

The k-tuple domination problem aims to find a k-tuple dominating function of G with the minimum labeling weight. Similarly, the total k-tuple domination problem seeks a total k-tuple dominating function with the minimum labeling weight.

Domination and total domination correspond to 1-tuple domination and total 1-tuple domination, respectively.

Definition 4. *Let k be a fixed positive integer. The labeling weight of a k-tuple dominating function or a total k-tuple dominating function f of a weighted graph (G, w) is defined as $\sum_{v \in V(G)} f(v) \cdot w(v)$. The k-tuple domination problem for a weighted graph (G, w) is to find a k-tuple dominating function with the minimum labeling weight. Similarly, the total k-tuple domination problem is to find a total k-tuple dominating function with the minimum labeling weight.*

Figure 3 illustrates a graph $G = (V, E)$ with six vertices. Table 4 provides the labeling weights for a 2-tuple dominating function f and a total 2-tuple dominating function g for both the unweighted graph G and the weighted graph (G, w).

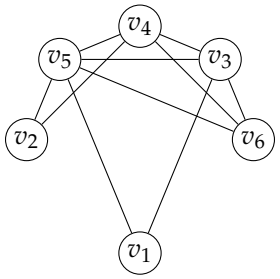


Figure 3. A graph with six vertices.

The function f assigns values as shown in the table, satisfying the 2-tuple domination condition for each vertex. For instance, vertex v_2 is dominated because $f(v_2) + f(v_4) + f(v_5) \geq 2$. The labeling weights differ for the unweighted and weighted versions due to the vertex weights $w(v)$.

Interestingly , g is also a 2-tuple dominating function of G. However, for the weighted graph (G, w), the labeling weight of g is smaller than that of f, demonstrating that g is more efficient in terms of weight minimization.

Table 4. Labeling weights for a 2-tuple dominating function f and a total $\{2\}$-tuple dominating function g in weighted and unweighted graphs.

Vertex v	Weight $w(v)$	$f(v)$	$g(v)$	$f(v) \cdot w(v)$	$g(v) \cdot w(v)$
v_1	2	1	0	2	0
v_2	3	0	0	0	0
v_3	1	0	1	0	1
v_4	4	1	1	4	4
v_5	2	1	1	2	2
v_6	2	0	0	0	0
Labeling Weight		3	3	8	7

 doi:10.20944/preprints202411.2117.v1

2.4. Strongly Chordal Graphs and Chordal Bipartite Graphs

A *chord* in a cycle of a graph is an edge that connects two non-consecutive vertices in the cycle. A *chordal* graph is a graph in which every cycle with four or more vertices has a chord. A *perfect elimination ordering* of a graph $G = (V, E)$ is an ordering $v_1, v_2, \ldots, v_n$ of the vertices such that for any indices $i < j < k$, if $(v_i, v_j) \in E$ and $(v_i, v_k) \in E$, then $(v_j, v_k) \in E$. Rose [48] proved that a graph is chordal if and only if it admits a perfect elimination ordering.

A chord (x_i, x_j) in a cycle C with vertices ordered as $x_1, x_2, \ldots, x_{2k}$ is called an *odd chord* if $|i - j|$ is odd. A graph G is a *strongly chordal* graph if it is chordal, and every cycle of $2k$ vertices in G, where $k \geq 3$, contains an odd chord. Figure 4 demonstrates a strongly chordal graph and the *Hajós graph*. The Hajós graph is chordal but not strongly chordal since the cycle $C = (u_1, u_2, \ldots, u_6)$ contains no odd chords.

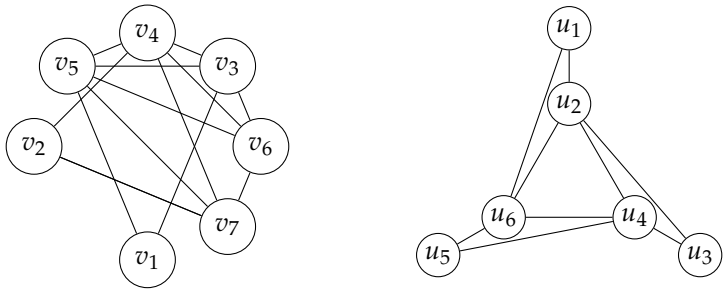


Figure 4. Two chordal graphs. (a) A strongly chordal graph. (b) The Hajós graph (right).

A *strong elimination ordering* $v_1, v_2, \ldots, v_n$ of a graph $G = (V, E)$ is a perfect elimination ordering such that for each $i < j$ and $k < \ell$, if $(v_i, v_\ell) \in E$, $(v_i, v_k) \in E$, and (v_k, v_j), then $(v_\ell, v_j) \in E$. Farber [49] proved that a graph is strongly chordal if and only if it admits a strong elimination ordering. Figure 4 shows a strongly chordal graph with a strong elimination ordering $v_1, v_2, \ldots, v_7$.

A *bipartite* graph is a graph $G = (X, Y, E)$ in which the vertex set $V(G)$ can be partitioned into two disjoint sets X and Y such that every edge in the graph connects a vertex in X to a vertex in Y, and no edge connects two vertices within the same set. In other words, there are no edges between vertices within X or within Y. Therefore, a bipartite graph does not have a cycle of an odd number of vertices. Figure 5 shows two graphs: The left one is bipartite, while the right one contains a cycle of three vertices and is therefore not bipartite.

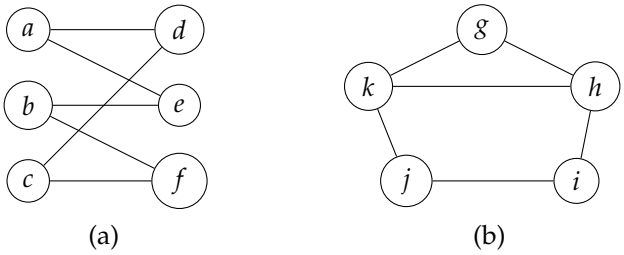


Figure 5. Two graphs. (a) A bipartite graph with six vertices. (b) A graph with a cycle of three vertices.

A *chordal bipartite* graph is a bipartite graph in which every cycle of more than four vertices contains a chord. Figure 6 shows two bipartite graphs: The left one is chordal bipartite, while the right one is not.

Let $G = (V, E)$ be a graph, and let $v_1, v_2, \ldots, v_n$ be an ordering of V. Let G_i be a subgraph of G induced by $\{v_i, v_{i+1}, \ldots, v_n\}$. A *weak elimination ordering* of G is an ordering $v_1, v_2, \ldots, v_n$ such that for any indices $i < j < k$, if $v_j, v_k \in N_{G_i}(v_i)$, then $N_{G_i}(v_j) \subseteq N_{G_i}(v_k)$.

 doi:10.20944/preprints202411.2117.v1

Uehara [50] showed that a graph is chordal bipartite if and only if it admits a weak elimination ordering. The ordering $v_1 = b_1, v_2 = a_2, v_3 = b_2, v_4 = a_3, v_5 = b_3, v_6 = a_1$ is a weak elimination ordering of the chordal bipartite graph in Figure 6(a).

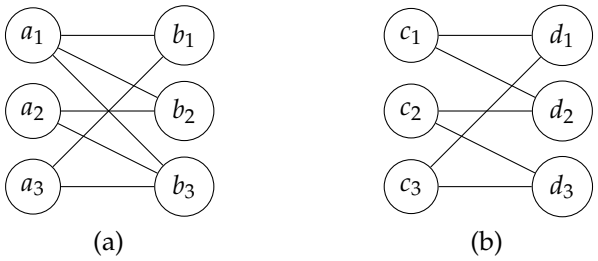


Figure 6. Two bipartite graphs. (a) A chordal bipartite graph. (b) A bipartite graph with an induced cycle $(c_1, d_2, c_2, d_3, c_3, d_1)$.

2.5. Proper Interval Graphs and Convex Bipartite Graphs

A graph $G = (V, E)$ is an *interval graph* if there exits an *interval model* on the real line such that each closed interval $I_v = [a_v, b_v]$ in the interval model corresponds to a vertex $v \in V$, and two vertices $u, v \in V$ are adjacent in G if and only if their corresponding intervals overlap, that is:

$$I_u \cap I_v \neq \emptyset.$$

Figure 7 illustrates an interval graph $G = (V, E)$, which is constructed from an interval model on the real line. The top part of the figure represents the intervals $[0.5, 2.5]$, $[2, 4.5]$, $[3.5, 5.5]$, and $[5, 7.5]$, labeled as vertices v_1, v_2, v_3, and v_4, respectively. An edge exists between two vertices if their corresponding intervals overlap. For example, the intervals $[0.5, 2.5]$ and $[2, 4.5]$ overlap, resulting in an edge between v_1 and v_2 in the graph. Similarly, v_2 and v_3 share an edge due to the overlap of their intervals, as do v_3 and v_4.

The lower part of the figure depicts the corresponding graph structure. The vertices v_1, v_2, v_3, and v_4 are connected by edges according to their interval overlaps, forming a path graph. .

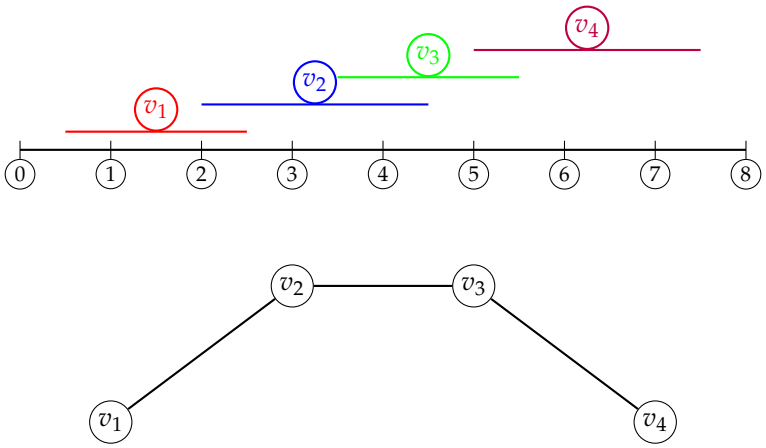


Figure 7. An interval graph constructed from the intervals $[0.5, 2.5]$, $[2, 4.5]$, $[3.5, 5.5]$, $[5, 7.5]$. The upper part shows the interval representation, and the lower part shows the corresponding graph.

An interval graph is called a *proper interval graph* if there exists an interval model for this interval graph such that for every pair of intervals $[a_u, b_u]$ and $[a_v, b_v]$ in the model, neither interval is strictly contained within the other, i.e., it is not true that $a_u \leq a_v$ and $b_v \leq b_u$. A *unit interval graph* is a special type of interval graph where all the intervals associated with the vertices have the same length. Actually, a unit interval graph G if and only if G is a proper interval graph [18]. They form the same class of graphs.

 doi:10.20944/preprints202411.2117.v1

Let $B = (X, Y, E)$ be a bipartite graph. An ordering P of X in B has the *adjacency property* if for each vertex $y \in Y$, $N_B(y)$ consists of vertices that are consecutive in the ordering P of X. A bipartite graph $B = (X, Y, E)$ is a *convex bipartite graph* if there is an ordering of X or Y satisfying the adjacency property.

The convex bipartite graph illustrated in Figure 8 consists of two disjoint sets of vertices, $U = \{u_1, u_2, u_3, u_4\}$ and $V = \{v_1, v_2, v_3, v_4\}$. The graph satisfies the adjacency property: for every vertex in U, the vertices of its neighbors in V are consecutive in the ordering. For instance, the neighbors of u_1 are v_1 and v_2 are consective in the ordering. Similarly, u_2 is adjacent to consecutive vertices v_2, v_3, and v_4. This property is consistent for all vertices in U.

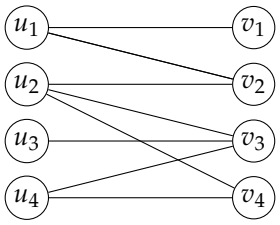


Figure 8. A convex bipartite graph.

2.6. Matrices and Vectors

A *matrix* is a rectangular array of numbers arranged in *rows* and *columns*. The horizontal arrangement of entries forms the rows, and the vertical arrangement forms the columns. An $m \times n$ matrix is a matrix with m rows and n columns. The individual numbers within the matrix are called *entries*. The entry in the i-th row and j-th column is called the (i, j)-entry.

Matrices are typically written within square brackets or parentheses; in this paper, they are written within parentheses. For example, a 3×4 matrix is written as

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}.$$

Let a_{ij} represent the entry in the i-th row and j-th column of an $m \times n$ matrix $\mathbf{A}$. The $m \times n$ matrix $\mathbf{A}$ can be denoted by $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{m \times n}$ and represented as follows:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix},$$

where $\mathbb{R}^{m \times n}$ represents the set of all matrices with m rows and n columns whose entries are real numbers. Formally, this is defined as

$$\mathbb{R}^{m \times n} = \{\mathbf{M} \mid \mathbf{M} \text{ is an } m \times n \text{ matrix and each entry of } \mathbf{M} \text{ is in } \mathbb{R}\}.$$

For any matrices $\mathbf{A} = (a_{ij})$ and $\mathbf{B} = (b_{ij})$ in $\mathbb{R}^{m \times n}$, we define the entrywise inequality $\mathbf{A} \leq \mathbf{B}$ to mean that $a_{ij} \leq b_{ij}$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$. For any scalar $k \in \mathbb{R}$, we write $\mathbf{A} \geq k$ to indicate that every entry of $\mathbf{A}$ is at least k. These definitions extend naturally to other similar cases involving entrywise comparisons. If each entry of $\mathbf{A}$ is k, then $\mathbf{A}$ is called a k-matrix.

Let $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{m \times n}$. The *transpose* of $\mathbf{A}$, denoted by $\mathbf{A}^T$, is an $n \times m$ matrix defined by $\mathbf{A}^T = (a_{ji}) \in \mathbb{R}^{n \times m}$. For a matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix},$$

 doi:10.20944/preprints202411.2117.v1

which is a 2×3 matrix, its transpose $\mathbf{A}^T$ is

$$\mathbf{A}^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix},$$

which is a 3×2 matrix.

A *submatrix* of a matrix $\mathbf{M}$ is obtained by removing any number of rows or columns from $\mathbf{M}$, including the possibility of removing none. Consequently, $\mathbf{M}$ is a submatrix of itself when no rows or columns are removed.

A *block* of matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a submatrix formed by rows $i, i+1, \ldots, j$ and columns $k, k+1, \ldots, \ell$. It is represented as:

$$\mathbf{A}(i:j, k:\ell),$$

where $1 \leq i \leq j \leq m$ and $1 \leq k \leq \ell \leq n$. The following matrix $\mathbf{A}$ consists of four blocks $\mathbf{A}_1 = \mathbf{A}(1:2, 1:2)$, $\mathbf{A}_2 = \mathbf{A}(1:2, 3:4)$, $\mathbf{A}_3 = \mathbf{A}(3:4, 1:2)$, and $\mathbf{A}_4 = \mathbf{A}(3:4, 3:4)$:

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{A}_2 \\ \mathbf{A}_3 & \mathbf{A}_4 \end{pmatrix}.$$

A *square matrix* is a matrix with the same number of rows and columns. The *identity matrix* is a square matrix $\mathbf{I} \in \mathbb{R}^{n \times n}$ such that an entry of $\mathbf{I}$ is 1 if its row index and column index are equal; otherwise, $x_{ij} = 0$. For example, the 3×3 identity matrix looks like this:

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

A *row vector* and a *column vector* are two types of vectors that differ in their orientation within a matrix or vector space. A row vector is a $1 \times n$ matrix represented as

$$\mathbf{v} = \begin{pmatrix} v_1 & v_2 & \ldots & v_n \end{pmatrix},$$

where $v_1, v_2, \ldots, v_n$ are the vector elements. A column vector is an $n \times 1$ matrix represented as

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix},$$

where $v_1, v_2, \ldots, v_n$ are the vector elements. Clearly, the transpose operation converts a row vector into a column vector and vice versa.

To ensure clarity and avoid ambiguity in notation, we adopt the following conventions: *Vectors* are represented by boldface upright lowercase letters, such as $\mathbf{v}$, $\mathbf{x}$, or $\mathbf{y}$. For example, $\mathbf{v} = (v_i) \in \mathbb{R}^n$ denotes a column vector with n entries. *Matrices* are represented by boldface upright uppercase letters, such as $\mathbf{A}$, $\mathbf{B}$, or $\mathbf{M}$, and entries of a matrix $\mathbf{A}$ are denoted by a_{ij} for the entry in the i-th row and j-th column. *Scalars*, by contrast, are represented by italicized lowercase letters, such as a, b, or c.

In general, when we refer to $\mathbf{v}$ as a vector, we mean that it is a column vector by default. If a row vector is intended, this will be explicitly indicated. If each entry of a vector is a constant k, the vector is called a k-vector.

 doi:10.20944/preprints202411.2117.v1

2.7. Totally Balanced, Totally Unimodular, and Greedy Matrices

A matrix $\mathbf{M}$ is called a $(0,1)$-*matrix* if every entry in $\mathbf{M}$ is either 1 or 0. In a matrix of numbers, the sum of all entries in a row is called the *row sum*, while the sum of all entries in a column is called the *column sum*. A $(0,1)$-matrix is said to be *totally balanced* if it does not contain any square submatrix satisfying all the following "forbidden conditions":

1. All row sums are equal to 2.
2. All column sums are equal to 2.
3. All columns are distinct.

Here is an example of a matrix that is *not* totally balanced:

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

To see why $\mathbf{A}$ is not totally balanced, consider the the submatrix $\mathbf{F} \in R^{3\times 3}$ obtained by removing the 4th and 5th rows and columns:

$$\mathbf{F} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

In $\mathbf{F}$, (1) all row sums are equal to 2, (2) all column sums are equal to 2, and (3) all columns are distinct. This submatrix satisfies the forbidden conditions, so $\mathbf{A}$ is not totally balanced. In contrast, the following matrix $\mathbf{B}$ is an example of a totally balanced matrix:

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

A $(0,1)$-matrix is *greedy* if it does not contain any of the following forbidden submatrices:

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

A greedy matrix is in *standard greedy form* [19] if it does not contain the following Γ-matrix as a submatrix:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

The matrix $\mathbf{L}$ below is a greedy matrix in standard greedy form. However, the matrix $\mathbf{Q}$ is a greedy matrix that is *not* in standard greedy form. This is because the submatrix of $\mathbf{Q}$ formed by deleting rows 3 and 4 and columns 3 and 4 is identical to the Γ-matrix.

$$\mathbf{L} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{Q} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

Let $\det(\mathbf{M})$ denote the determinant of a matrix $\mathbf{M}$. The matrix M is *totally unimodular* if $\det(\mathbf{M}') \in \{-1, 0, 1\}$ for every square submatrix $\mathbf{M}'$ of $\mathbf{M}$.

 doi:10.20944/preprints202411.2117.v1

Consider the matrices **A** and **B**:

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}.$$

The matrices **A** and **B** illustrate examples of a totally unimodular matrix and a matrix that is not totally unimodular, respectively. Matrix **A** is totally unimodular because the determinant of every square submatrix of **A** is in $\{-1, 0, 1\}$. In contrast, matrix **B** is not totally unimodular. For example, $\det(\mathbf{B})$ is 2. Thus, **B** fails to satisfy the defining condition of total unimodularity. This comparison underscores the stringent conditions a matrix must satisfy to be classified as totally unimodular.

Furthermore, a matrix has the *consecutive ones property* if its rows can be permuted in such a way that, in every column, all the 1s appear consecutively. This property is particularly relevant in applications involving binary matrices and is a useful tool for analyzing matrix structures.

The following theorem reveals a connection between the consecutive ones property and total unimodularity.

Theorem 1 ([45]). *If a $(0,1)$-matrix has the consecutive ones property for columns, then it is totally unimodular.*

2.8. Linear and Integer Linear Programming

Linear programs are optimization problems that maximize or minimize a linear objective function, subject to linear constraints. A *minimization linear program* seeks to minimize the objective function, while a *maximization linear program* aims to maximize it.

Linear programming is the field of study and methodology focused on formulating, analyzing, and solving linear programs. It encompasses the theories, algorithms, and techniques developed for solving linear programs. In essence, linear programming refers to the process and theory, while a linear program is an individual problem instance.

Formulating a minimize linear program requires the following inputs: n real numbers $c_1, c_2, \ldots, c_n$ (coefficients of the objective function), m real numbers $b_1, b_2, \ldots, b_m$ (resource limits), and $m \times n$ coefficients a_{ij} for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$ (constraint coefficients). The goal is to find values for the decision variables $x_1, x_2, \ldots, x_n$ that minimize the objective function, subject to all constraints. This formulation is given by:

$$\text{minimize} \quad \sum_{j=1}^{n} c_j x_j \tag{P1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad \text{for } i = 1, 2, \ldots, m \tag{P2}$$

$$x_j \geq 0 \quad \text{for } j = 1, 2, \ldots, n. \tag{P3}$$

In this formulation, the expression in (P1) is the *objective function*; the variables $x_1, x_2, \ldots, x_n$ are called *decision variables*; and the inequalities in (P2) and (P3) together form the *constraints*. Specifically, the n inequalities in (P3) are *nonnegativity constraints*, requiring each x_j to be nonnegative.

In mathematical optimization, *duality* provides two perspectives on an optimization problem: the *primal* problem and its corresponding *dual*. When the primal is a minimization problem, its dual is a maximization problem, and conversely, when the primal is a maximization problem, the dual will be a minimization problem. Thus, for a pair of primal and dual problems, one is a maximization problem, and the other is a minimization problem.

Weak duality states that, for any feasible solutions to a pair of primal and dual problems, the value of the maximization problem is always less than or equal to the value of the minimization problem.

 doi:10.20944/preprints202411.2117.v1

Strong duality further asserts that, at optimality, the optimal value of the primal problem is equal to the optimal value of its dual.

For any linear program considered as the *primal*, there exists a corresponding *dual* linear program. Each constraint in the primal corresponds to a variable in the dual, and each decision variable in the primal corresponds to a constraint in the dual. The dual of the minimization linear program in equations P1–P3 is formulated as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{m} b_i y_i \\ \text{subject to} \quad & \sum_{i=1}^{m} a_{ij} y_i \leq c_j \quad \text{for } j = 1, 2, \ldots, n \\ & y_i \geq 0 \quad \text{for } i = 1, 2, \ldots, m. \end{aligned}$$

In this dual problem, $y_1, y_2, \ldots, y_m$ are the dual variables associated with each primal constraint; $b_1, b_2, \ldots, b_m$ are the coefficients of the dual objective function; and constraints in the dual correspond to the primal variables, and $c_1, c_2, \ldots, c_n$ values become the bounds in the dual constraints.

For a minimization linear problem, the primal can be written in matrix form as

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq 0, \end{aligned}$$

where $\mathbf{c} = (c_j) \in \mathbb{R}^n$, $\mathbf{x} = (x_j) \in \mathbb{R}^n$, $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{m \times n}$, and $\mathbf{b} = (b_i) \in \mathbb{R}^m$. Its dual can be expressed as

$$\begin{aligned} \text{maximize} \quad & \mathbf{b}^T \mathbf{y} \\ \text{subject to} \quad & \mathbf{A}^T \mathbf{y} \leq \mathbf{c} \\ & \mathbf{y} \geq 0, \end{aligned}$$

where, $\mathbf{y} = (y_i) \in \mathbb{R}^m$; $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{c}$ are as defined in the primal formulation.

In many cases, capturing the essence of a linear program is best achieved without excessive notation or detail. To highlight the fundamental structure clearly and concisely, we adopt forms such as

$$\min\{\mathbf{c}^T \mathbf{x} \mid \mathbf{x} \geq 0,\ \mathbf{A}\mathbf{x} \geq \mathbf{b}\} \quad \text{and} \quad \max\{\mathbf{c}^T \mathbf{x} \mid \mathbf{x} \geq 0,\ \mathbf{A}\mathbf{x} \leq \mathbf{b}\},$$

to facilitate analysis, communication, and efficient solution of linear programs.

A solution to an optimization problem is a column vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$, often called a *point* because each solution corresponds to a specific set of values for the variables, which can be represented as a single point in geometric space. In other words, each feasible solution is associated with a unique location in this space. A solution that satisfies all constraints is known as a *feasible solution*, whereas a solution that fails to satisfy at least one constraint is an *infeasible solution*. The set of points that satisfy all the constraints is referred to as the *feasible region*. In linear programming, the feasible region is called a *polyhedron*, as it is the intersection of linear constraints. A polyhedron is often denoted by P, such as $P = \{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} \geq \mathbf{b}\}$, representing the set of all points $\mathbf{x}$ that meet the specified constraints.

In an optimization problem, a constraint is called *active* at a given solution point if the solution causes the constraint to hold as an equality, effectively "binding" the solution at the boundary defined by the constraint. An *extreme point* of a polyhedron occurs where several of the constraints in the problem are active, often as many as the dimension of the space.

The *fundamental theorem of linear programming* ([51]) states that if a linear program has an optimal solution, at least one optimal solution will be located at an extreme point of the polyhedron. This result

 doi:10.20944/preprints202411.2117.v1

is foundational because it allows optimization algorithms like the simplex method to focus on the extreme points of the feasible region, significantly reducing the computational effort required to find an optimal solution. If every extreme point of a polyhedron consists of integer values for all variables, then the polyhedron is said to be *integral*.

Theorem 2 establishes that linear programs satisfy strong duality.

Theorem 2 ([44]).

$$\min\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \geq 0,\ \mathbf{A}\mathbf{x} \geq \mathbf{b}\} = \max\{\mathbf{b}^T\mathbf{y} \mid \mathbf{y} \geq 0,\ \mathbf{A}^T\mathbf{y} \leq \mathbf{c}\}.$$

An *integer linear program* is an optimization model similar to a linear program but includes an additional constraint that requires all decision variables to be integers. This distinction substantially impacts both the complexity and solution methods for these problems. While linear programs can be solved in polynomial time [52], integer linear programs are NP-complete [53], making them more computationally challenging.

Integer linear programs satisfy *weak duality* but do not always satisfy *strong duality*, meaning the optimal values of an integer linear program and its dual may differ.

To solve integer linear programs, advanced methods such as branch-and-bound, branch-and-cut, and cutting planes are often employed. These methods frequently use *linear relaxations*, which remove the integer constraints to produce a continuous solution, providing useful bounds on the solution to an integer linear program. In certain cases, the optimal objective value of an integer linear program matches that of its linear relaxation, as illustrated by Theorem 3.

Theorem 3 ([54]). *If* $\mathbf{A}$, $\mathbf{b}$, *and* $\mathbf{c}$ *all have integer entries, with at least one of* $\mathbf{b}$ *or* $\mathbf{c}$ *as a constant vector, and* $\mathbf{A}$ *is totally balanced, then the optimal objective values of the integer program*

$$\max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \in \mathbb{Z} \text{ and } \mathbf{x} \geq 0\}$$

and its linear relaxation are the same.

3. $\{k\}$-Domination in Weighted Strongly Chordal Graphs

In this section, we demonstrate that the $\{k\}$-domination problem in weighted strongly chordal graphs with n vertices and m edges can be solved in $\mathcal{O}(n+m)$ time. The solution follows a four-step framework, as outlined below:

(1) **Modeling (Section 3.1):** The $\{k\}$-domination problem is formulated as an integer linear programming task using matrix representations, particularly adjacency matrices. This formulation establishes the theoretical foundation for the algorithmic approaches described in subsequent sections. We aim to solve the integer linear program by its relaxation.

(2) **Primal and Dual Algorithms by Hoffman et al. (Section 3.2):** We introduce the primal and dual algorithms proposed by Hoffman et al. [19], which solve the linear program:

$$\min\{\mathbf{c}^T\mathbf{x} + \mathbf{d}^T\mathbf{z} \mid \mathbf{A}\mathbf{x} + \mathbf{z} \geq \mathbf{b}, \mathbf{x} \geq 0, \mathbf{z} \geq 0\}.$$

These algorithms provide a robust framework for solving linear programming problems, forming the basis for adapting solutions to the $\{k\}$-domination problem.

(3) **Refined Algorithms for Weighted Strongly Chordal Graphs (Section 3.3):** Building on Hoffman et al.'s algorithms, this section introduces refinements tailored to the structural properties of weighted strongly chordal graphs. These refinements reduce the computational complexity to $\mathcal{O}(n^2)$, representing an intermediate step toward the final optimized solution.

(4) **Optimized Algorithms with Enhanced Data Structures (Section 3.4):** By integrating advanced data structures, this section further reduces the overall time complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n+m)$.

 doi:10.20944/preprints202411.2117.v1

This optimization capitalizes on the sparsity and adjacency structure of strongly chordal graphs, achieving linear-time performance relative to the graph's size.

3.1. Modeling

We start by presenting Lemma 1 to concentrate exclusively on non-negative vertex weights ($\mathbb{R}_{\geq 0}$). This simplification allows us to assume that all weighted graphs have non-negative vertex weights.

Lemma 1. *Let w be a vertex-weight function of a graph $G = (V, E)$, and let $\gamma_k(G, w)$ represent the the minimum labeling weight of a $\{k\}$-dominating function for (G, w). Define $S = \{v \mid v \in V, w(v) < 0\}$ and let w' be a vertex-weight function such that $w'(v) = \max\{w(v), 0\}$ for each $v \in V$. Then, $\gamma_k(G, w) = \gamma_k(G, w') + k\sum_{v\in S} w(v)$.*

Proof. Let f be a $\{k\}$-dominating function for (G, w) with minimum weight, so $\gamma_k(G, w) = \sum_{v\in V} w(v)f(v)$. Define a function f' such that $f'(v) = k$ for $v \in S$ and $f'(v) = f(v)$ for $v \in V \setminus S$. Since f' is also a $\{k\}$-dominating function of G and $w'(v) = 0$ for $v \in S$, we obtain

$$\begin{aligned}\gamma_k(G, w') &\leq \sum_{v\in V} w'(v)f'(v) = \sum_{v\in V\setminus S} w(v)f(v)\\ &= \sum_{v\in V} w(v)f(v) - \sum_{v\in S} w(v)f(v)\\ &\leq \gamma_k(G, w) - k\sum_{v\in S} w(v).\end{aligned}$$

Conversely, let h be a $\{k\}$-dominating function for (G, w') with the minimum labeling weight, so

$$\gamma_k(G, w') = \sum_{v\in V} w'(v)h(v) = \sum_{v\in V\setminus S} w'(v)h(v).$$

Define f such that $f(v) = k$ for $v \in S$ and $f(v) = h(v)$ for $v \in V \setminus S$. Clearly, f is a $\{k\}$-dominating function of G, and $w'(v) = 0$ for $v \in S$. We have

$$\begin{aligned}\gamma_k(G, w) &\leq \sum_{v\in V} w(v)f(v) = \sum_{v\in V\setminus S} w(v)f(v) + k\sum_{v\in S} w(v)\\ &= \sum_{v\in V\setminus S} w'(v)h(v) + k\sum_{v\in S} w(v)\\ &= \gamma_k(G, w') + k\sum_{v\in S} w(v).\end{aligned}$$

This completes the proof. □

Let (G, w) be a weighted graph with $V(G) = \{v_1, v_2 \ldots, v_n\}$. We associate a variable x_j with each $v_j \in V(G)$ and require that $x_j \in \{0, 1, \ldots, k\}$ for $j = 1, 2, \ldots, n$. Let $w_j = w(v_j)$ for $1 \leq i \leq n$. The $\{k\}$-domination problem for the weighted graph G is formulated as the following integer linear program $IP_k(G, w)$:

$$\begin{aligned}\text{minimize} \quad & \sum_{j=1}^{n} w_j x_j\\ \text{subject to} \quad & \sum_{v_j\in N_G[v_i]} x_j \geq k \quad \text{for } i = 1, 2, \ldots, n\\ & x_j \in \{0, 1, \ldots, k\} \quad \text{for } j = 1, 2, \ldots, n.\end{aligned}$$

Lemma 2. *Let $\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_n^*)$ be an optimal solution to $IP_k(G, w)$. Then, $x_j^* \leq k$ for all $j = 1, 2, \ldots, n$.*

 doi:10.20944/preprints202411.2117.v1

Proof. Clearly, $\sum_{j=1}^{n} w_j x_j^*$ is the minimized and $x_j \geq 0$ for $j = 1, 2, \ldots, n$. Assume that there exists an x_i^* in $\mathbf{x}^*$ such that $x_i^* > k$. For any constraint involving x_i^*, the constraint remains satisfied if x_i^* is replaced with k. Consequently, the resulting objective value is less than $\sum_{j=1}^{n} w_j x_j^*$, leading to a contradiction. Therefore, the lemma holds. □

By Lemma 2, we can reformulate $IP_k(G, w)$ as follows:

$$
\begin{aligned}
& IP_k(G, w): \\
& \text{minimize} \quad \sum_{j=1}^{n} w_j x_j \\
& \text{subject to} \quad \sum_{v_j \in N_G[v_i]} x_j \geq k \quad \text{for } i = 1, 2, \ldots, n \\
& \qquad\qquad x_j \geq 0 \text{ and } x_j \in \mathbb{Z} \quad \text{for } i = 1, 2, \ldots, n.
\end{aligned}
$$

In graph theory, the *neighborhood matrix* and *the closed neighborhood matrix* are two distinct representations of the adjacency relationships in a graph H with vertices $V(H) = \{v_1, v_2, \ldots, v_n\}$.

- The neighborhood matrix of G is a matrix $\mathbf{N} = (n_{ij}) \in \mathbb{R}^{n \times n}$ such that $n_{ij} = 1$ if $v_j \in N_H(v_i)$; otherwise, $n_{ij} = 0$.
- The closed neighborhood matrix of G is a matrix $\mathbf{M} = (m_{ij}) \in \mathbb{R}^{n \times n}$ such that $m_{ij} = 1$ if $v_j \in N_H[v_i]$; otherwise, $m_{ij} = 0$.

Let $\mathbf{M} = (m_{ij}) \in \mathbb{R}^{n \times n}$ be the closed neighborhood matrix of the weighted graph G with $\mathbf{w} = (w_j) \in \mathbb{R}^n_{\geq 0}$, $\mathbf{x} = (x_j) \in \mathbb{R}^n$, and $\mathbf{p} = (p_i) \in \mathbb{R}^n$, where $p_i = k$ for $1 \leq i \leq n$.

We present the linear program $LP_k(G, w)$ and its dual $LP_k^d(G, w)$ using the closed neighborhood matrix $\mathbf{M}$.

Primal ($LP_k(G, w)$):		Dual ($LP_k^d(G, w)$):	
minimize	$\mathbf{w}^T \mathbf{x}$	maximize	$\mathbf{p}^T \mathbf{y}$
subject to	$\mathbf{M}\mathbf{x} \geq \mathbf{p}$	subject to	$\mathbf{M}^T \mathbf{y} \leq \mathbf{w}$
	$\mathbf{x} \geq 0$.		$\mathbf{y} \geq 0$.

Let IP_{opt} and LP_{opt} denote the optimal objective values of an integer linear program IP and its linear relaxation LP, respectively. The polyhedron for IP is a subset of the polyhedron for LP, since IP includes the additional constraint that variables must be integers. Therefore, the optimal value of LP, minimized over a larger polyhedron, cannot be greater the optimal value of IP. Hence, $LP_{\text{opt}} \leq IP_{\text{opt}}$.

In subsequent sections, we aim to demonstrate that for weighted strongly chordal graphs, the optimal value of $IP_k(G, w)$ is equal to the optimal value of its linear relaxation $LP_k(G, w)$. This allows us to solve this integer linear program by obtaining an integral solution directly from its linear relaxation.

3.2. Primal and Dual Algorithms by Hoffman et al.

Let $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{m \times n}$ be a greedy matrix in standard form, with vectors $\mathbf{x} = (x_j) \in \mathbb{R}^n$, $\mathbf{y} = (y_i) \in \mathbb{R}^m$, and $\mathbf{z} = (z_i) \in \mathbb{R}^m$ as variables. Let $\mathbf{c} = (c_j) \in \mathbb{R}^n$, $\mathbf{b} = (b_i) \in \mathbb{R}^m$, and $\mathbf{d} = (d_i) \in \mathbb{R}^m$ be constant vectors with $\mathbf{c} \geq 0$, $\mathbf{d} \geq 0$, and $b_1 \geq b_2 \geq \cdots \geq b_m \geq 0$. The primal linear program P is defined as:

$$
\begin{aligned}
& \text{minimize} \quad \mathbf{c}^T \mathbf{x} + \mathbf{d}^T \mathbf{z} \\
& \text{subject to} \quad \mathbf{A}\mathbf{x} + \mathbf{z} \geq \mathbf{b} \\
& \qquad\qquad \mathbf{x} \geq 0, \quad \mathbf{z} \geq 0.
\end{aligned}
$$

 doi:10.20944/preprints202411.2117.v1

The dual program D is formulated as:

$$\begin{aligned} \text{maximize} \quad & \mathbf{b}^T\mathbf{y} \\ \text{subject to} \quad & \mathbf{A}^T\mathbf{y} \leq \mathbf{c} \\ & \mathbf{y} \geq 0, \quad \mathbf{y} \leq \mathbf{d}. \end{aligned}$$

Hoffman et al. [19] introduced two greedy algorithms to solve P and its dual D in polynomial time, and constructed an integer optimal solution for P. These algorithms are presented in Algorithms 1 and 2.

Algorithm 1 provides an integer optimal solution for D. The dual program D contains n constraints and m decision variables $y_1, y_2, \ldots, y_m$. A constraint j is defined as *tight* if

$$\sum_{i=1}^{m} a_{ij} y_i = c_j.$$

Let $J = \{j \mid j \text{ is tight}\}$ and $\alpha(j) = \max\{i \mid a_{ij} = 1, y_i > 0, \sum_{i=1}^{m} a_{ij} y_i = c_j\}$. Algorithm 1 determines each variable y_i in increasing order of i and takes the largest feasible value. It also computes J and $\alpha(j)$ for use in Algorithm 2.

Algorithm 1: Dual Solution for Program D with Greedy Matrices

```
J ← ∅; ĉ ← c;
for i ← 1 to m do
    y_i ← min{d_i, min{ĉ_j | a_ij = 1}};
    if y_i > 0 then
        if y_i = ĉ_j for some j then
            Choose the largest j satisfying the condition;
            J ← J ∪ {j};
            α(j) ← i;
        Update ĉ_j ← ĉ_j − y_i for all j such that a_ij = 1;
```

Algorithm 2 computes the primal solution corresponding to the dual solution by iteratively adjusting x_j values.

Algorithm 2: Primal Solution for Program P Corresponding to Dual Solution

```
b̂ ← b; x_j ← 0 for all j ∉ J;
while J ≠ ∅ do
    Let k be the last column of J;
    Set x_k ← b̂_{α(k)};
    Update b̂_i ← b̂_i − x_k for all i such that a_ik = 1;
    Remove k from J;
for i ← 1 to m do
    z_i ← max(0, b̂_i);
```

Theorem 4 ([19]). *The dual program D is solved by Algorithm 1 for all $\mathbf{c} \geq 0, \mathbf{d} \geq 0$, and $b_1 \geq b_2 \geq \cdots \geq b_m \geq 0$ if and only if A is greedy in standard greedy form. Further, Algorithm 2 constructs an integer optimal solution to the primal problem P. Both algorithms run in $\mathcal{O}(mn)$ time.*

 doi:10.20944/preprints202411.2117.v1

3.3. Refined Algorithms for Weighted Strongly Chordal Graphs

We present Lemma 3 as a fundamental observation linking greedy matrices in standard greedy form to the closed neighborhood matrices of strongly chordal graphs with strong elimination orderings.

Lemma 3. *If the vertices of a strongly chordal graph G are ordered by its strong elimination ordering, then the closed neighborhood matrix of G is a greedy matrix in standard greedy form.*

Proof. We start by proving the following claim.

Claim 1. *A greedy matrix in standard greedy form is precisely a totally balanced matrix that contains no submatrix identical to the Γ-matrix.*

Let S_1 represent the set of all greedy matrices in standard greedy form, and let S_2 represent the set of all totally balanced matrices that exclude any submatrix identical to the Γ-matrix. To prove the claim, we will show that $S_1 = S_2$.

According to Hoffman et al. [19], every greedy matrix is totally balanced. This establishes that $S_1 \subseteq S_2$.

By definition, a greedy matrix is a $(0,1)$-matrix that does not contain either of the following forbidden submatrices:

$$F_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \text{and} \quad F_2 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

It is clear that a totally balanced matrix that excludes any submatrix identical to the Γ-matrix must also exclude F_1 and F_2 as submatrices. Therefore, any matrix in S_2 must also be in S_1, giving us $S_2 \subseteq S_1$.

Since we have both $S_1 \subseteq S_2$ and $S_2 \subseteq S_1$, it follows that $S_1 = S_2$. This completes the proof of the claim.

Farber [49] showed that if the vertices of a strongly chordal graph G are ordered by its strong elimination ordering, then the closed neighborhood matrix of G is totally balanced and excludes any submatrix identical to the Γ-matrix.

Thus, the lemma hols. □

For clarity and ease of reference, we present the formulations of P, D, $LP_k(G,w)$, and $LP_k^d(G,w)$ together below.

Primal (P):		Dual (D):	
minimize	$\mathbf{c}^T\mathbf{x} + \mathbf{d}^T\mathbf{z}$	maximize	$\mathbf{b}^T\mathbf{y}$
subject to	$\mathbf{A}\mathbf{x} + \mathbf{z} \geq \mathbf{b}$	subject to	$\mathbf{A}^T\mathbf{y} \leq \mathbf{c}$
	$\mathbf{x} \geq 0, \quad \mathbf{z} \geq 0.$		$\mathbf{y} \geq 0, \quad \mathbf{y} \leq \mathbf{d}.$
Primal ($LP_k(G,w)$):		Dual ($LP_k^d(G,w)$):	
minimize	$\mathbf{w}^T\mathbf{x}$	maximize	$\mathbf{p}^T\mathbf{y}$
subject to	$\mathbf{M}\mathbf{x} \geq \mathbf{p}$	subject to	$\mathbf{M}^T\mathbf{y} \leq \mathbf{w}$
	$\mathbf{x} \geq 0.$		$\mathbf{y} \geq 0.$

Theorem 5. *For weighted strongly chordal graphs (G,w) with n vertices arranged by strong elimination ordering, the $\{k\}$-domination problem can be solved in $\mathcal{O}(n^2)$ time.*

Proof. Lemma 3 allows us to observe that for a weighted strongly chordal graph (G,w) with n vertices arranged in strong elimination ordering, the closed neighborhood matrix $\mathbf{M}$ of G is a greedy matrix in standard greedy form. Additionally, in the linear program $LP_k(G,w)$, we have $\mathbf{w} \geq 0$, and the vector

 doi:10.20944/preprints202411.2117.v1

$\mathbf{p}$ is constant with entries equal to k. Consequently, the programs $LP_k(G,w)$ and its dual $LP_k^d(G,w)$ for the weighted strongly chordal graph (G,w) are specific instances of the programs P and D, where $\mathbf{z}=0$ and the vector $\mathbf{d}$ is absent.

This connection enables us to modify and simplify Hoffman et al.'s original algorithms, Algorithms 1 and 2, to yield Algorithms 3 and 4. We present Algorithms 3 and 4 below with the necessary line-by-line transformations.

To modify and simplify Algorithm 1 into Algorithm 3, the following line-by-line adjustments (highlighted in blue) were made:

1. Line 1: Replace $\hat{\mathbf{c}}$ with $\hat{\mathbf{w}}$ and $\mathbf{c}$ with $\mathbf{w}$.
2. Line 2: Adjust the loop range from m to n.
3. Line 3: Simplify the min calculation from $\min\{d_i, \min\{\hat{c}_j \mid a_{ij}=1\}\}$ to $\min\{\hat{w}_j \mid m_{ij}=1\}$.
4. Lines 5 and 6: Replace the original lines with the simplified expression $j = \max\{j \mid y_i = \hat{w}_j, m_{ij}=1\}$. Note that in the simplified algorithm, d_i is unnecessary in Line 3. Therefore, if $y_i > 0$, then y_i from $\min\{\hat{w}_j \mid m_{ij}=1\}$ must equal $\hat{w}_j$ for some j. Thus, choosing the largest j such that $y_i = \hat{w}_j$ can be expressed equivalently as $j = \max\{j \mid y_i = \hat{w}_j, m_{ij}=1\}$.
5. Line 9: Substitute each $\hat{c}_j$ with $\hat{w}_j$ and a_{ij} with m_{ij} in the update statement.

Algorithm 3: Simplified Algorithm for $LP_k^d(G,w)$

1: $J \leftarrow \emptyset$; $\hat{\mathbf{w}} \leftarrow \mathbf{w}$;
2: **for** $i \leftarrow 1$ **to** n **do**
3: $\quad y_i \leftarrow \min\{\hat{w}_j \mid m_{ij}=1\}$;
4: $\quad$ **if** $y_i > 0$ **then**
5: $\quad\quad j = \max\{j | y_i = \hat{w}_j, m_{ij}=1\}$;
6: $\quad\quad J \leftarrow J \cup \{j\}$;
7: $\quad\quad \alpha(j) \leftarrow i$;
8: $\quad$ Update $\hat{w}_j \leftarrow \hat{w}_j - y_i$ for all j such that $m_{ij}=1$;

To modify and simplify Algorithm 2 into Algorithm 4, we made the following adjustments (highlighted in red):

1. Line 1: Change $\hat{\mathbf{b}}$ to $\hat{\mathbf{p}}$. Replace $\mathbf{b}$ with $\mathbf{p}$.
2. Line 4: Adjust $\hat{b}_{\alpha(k)}$ to $\hat{p}_{\alpha(k)}$.
3. Line 5: Replace $\hat{b}_i$ with $\hat{p}_i$. Replace a_{ik} with m_{ik}.
4. Remove Lines 7 and 8, since they are unnecessary in the simplified algorithm for $LP_k(G,w)$.

Algorithm 4: Simplified Algorithm for $LP_k(G,w)$ with Totally Balanced Matrices

1: $\hat{\mathbf{p}} \leftarrow \mathbf{p}$; $x_j \leftarrow 0$ for all $j \notin J$;
2: **while** $J \neq \emptyset$ **do**
3: $\quad$ Let k be the last column of J;
4: $\quad$ Set $x_k \leftarrow \hat{p}_{\alpha(k)}$;
5: $\quad$ Update $\hat{p}_i \leftarrow \hat{p}_i - x_k$ for all i such that $m_{ik}=1$;
6: $\quad$ Remove k from J;

By Theorem 4, the linear program $LP_k(G,w)$ and its dual $LP_k^d(,w)$ are solvable in $\mathcal{O}(n^2)$ time for weighted strongly chordal graphs G with vertices in strong elimination ordering, and Algorithms 3 and 4 can provide integer optimal solutions for $LP_k(G,w)$ and $LP_k^d(G,w)$. Thus, the $\{k\}$-domination problem for weighted strongly chordal graphs can be solved in $\mathcal{O}(n^2)$ time. □

 doi:10.20944/preprints202411.2117.v1

3.4. Optimized Algorithms with Enhanced Data Structures

In the previous section, we established that the linear program $LP_k(G, w)$ and its dual are special cases of the problems tackled by Hoffman et al.'s algorithms. This enables us to apply their method directly to solve these programs in $\mathcal{O}(n^2)$ time. Hoffman et al.'s algorithms are robustly constructed around the relationships between totally balanced matrices, greedy matrices, and linear programs, using the strengths of matrix structures and concepts from linear and integer programming. In contrast, the closed neighborhood matrices $\mathbf{M}$ in $LP_k(G, w)$ are closely tied to graph adjacency structures.

The adjacency-list representation, as discussed in Section 2.1, plays a crucial role in enhancing the efficiency of Algorithms 3 and 4. By incorporating optimized data structures for critical steps in these algorithms, we improve the overall time complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n + m)$, where n represents the number of vertices, and m represents the number of edges in a graph. Since the maximum number of edges in a graph is $\binom{n}{2} = \mathcal{O}(n^2)$, this improvement makes the algorithms particularly effective for sparse strongly chordal graphs when $m \ll n^2$. Theorem 6 demonstrates how to obtain the desired time complexity.

Theorem 6. *For weighted strongly chordal graphs (G, w) with n vertices arranged by strong elimination ordering, the $\{k\}$-domination problem can be solved in $\mathcal{O}(n + m)$ time, where m is the number of edges in G.*

Proof. We begin with demonstrating how each line of Algorithm 3 relates to the adjacency list and other optimized data structures and the running time of each line based on the adjacency list representation and the number of vertices (n) and edges (m):

Algorithm 3: Simplified Algorithm for $LP_k^d(G, w)$

1: $J \leftarrow \emptyset$; $\hat{\mathbf{w}} \leftarrow \mathbf{w}$;
2: **for** $i \leftarrow 1$ **to** n **do**
3: $\quad y_i \leftarrow \min\{\hat{w}_j \mid m_{ij} = 1\}$;
4: $\quad$ **if** $y_i > 0$ **then**
5: $\quad\quad j = \max\{j | y_i = \hat{w}_j, m_{ij} = 1\}$;
6: $\quad\quad J \leftarrow J \cup \{j\}$;
7: $\quad\quad \alpha(j) \leftarrow i$;
8: $\quad$ Update $\hat{w}_j \leftarrow \hat{w}_j - y_i$ for all j such that $m_{ij} = 1$;

1. **Line 1**:

   This line initializes the set J to keep track of selected indices j and makes a copy of the vector $\mathbf{w}$ as $\hat{\mathbf{w}}$. This step does not directly involve the adjacency list, as it's a basic initialization of variables. We implement J as an array with n entries and initialize the array with each entry set to zero. This array structure allows us to store specific information about each element efficiently, and each entry can be accessed, updated, or retrieved in $\mathcal{O}(1)$ time. The initialization step takes $\mathcal{O}(n)$ time.
2. **Line 2**:

   This loop iterates over each index i, where n corresponds to the number of constraints equal to the number of vertices. It executes n times in total. For each iteration, it processes Lines 3–8.

   (a) **Line 3**: For each i, the algorithm assigns y_i the minimum value among the entries $\hat{w}_j$ for all j such that $m_{ij} = 1$. Clearly, $m_{ij} = 1$ means $v_j \in N_G[v_i]$. Using the adjacency list, the algorithm can directly access all neighbors of v_i, including v_i itself, in $\mathcal{O}(deg_G(v_i) + 1)$ time.

   (b) **Line 4**:

   This line checks whether the calculated y_i value is greater than 0. It can be done in $\mathcal{O}(1)$ time

 doi:10.20944/preprints202411.2117.v1

(c) **Lines 5-7**: This block finds the largest j for i such that $y_i = \hat{w}_j$ and $m_{ij} = 1$.

The adjacency list helps here by providing access to each index j of $v_j \in N_G[v_i]$, allowing the algorithm to identify all possible values of j where $y_i = \hat{w}_j$. Once the appropriate j is found, J and $\alpha(j)$ are updated. Since J is an array, the operation $J \cup \{j\}$ is translated to setting $J[j] = 1$. Conversely, J does not contain some k if $J[k] = 0$. We also implement the operation $\alpha(j) \leftarrow i$ by an array. The operation $\alpha(j) \leftarrow i$ is translated to setting $\alpha[j] = i$.

Hence, the steps take $\mathcal{O}(deg_G(v_i) + 1)$ time.

(d) **Line 8**:

This line updates $\hat{w}_j$ for each index j of vertices $v_j \in N_G[v_i]$, subtracting y_i from $\hat{w}_j$. Using the adjacency list, the algorithm can efficiently locate each j where $m_{ij} = 1$ and apply the update only to those specific j values for vertices $v_j \in N_G[v_i]$.

The overall time complexity depends on the sum of all neighbor operations across vertices, yielding

$$\mathcal{O}\left(n + \sum_{i=1}^{n}(deg_G(v_i) + 1)\right) = \mathcal{O}(n + m),$$

where n is the number of vertices and m is the number of edges.

We next demonstrate how each line of Algorithm 4 relates to the adjacency list and other optimized data structures and the running time of each line based on the adjacency list representation and the number of vertices (n) and edges (m):

Algorithm 4: Simplified Algorithm for $LP_k(G, w)$ with Totally Balanced Matrices

1: $\hat{\mathbf{p}} \leftarrow \mathbf{p}$; $x_j \leftarrow 0$ for all $j \notin J$;
2: **while** $J \neq \emptyset$ **do**
3: Let k be the last column of J;
4: Set $x_k \leftarrow \hat{p}_{\alpha(k)}$;
5: Update $\hat{p}_i \leftarrow \hat{p}_i - x_k$ for all i such that $m_{ik} = 1$;
6: Remove k from J;

1. Line 1: This line initializes $\hat{\mathbf{p}}$ as a copy of $\mathbf{p}$ and sets $x_j = 0$ for all $j \notin J$.

   To set $x_j = 0$ for all $j \notin J$, we implement a stack $\hat{J}$ and set it to be empty. Then, we visit each entry $J[j]$ for $1 \leq j \leq n$, set $x_j = 0$ if $J[j] = 0$, and push j into the stack $\hat{J}$ if $J[j] \neq 0$. After visiting all entries of J, we delete J and rename $\hat{J}$ with J. All operations can be done in $\mathcal{O}(n)$ time. Since $\mathbf{p}$ also has n entries, copying $\mathbf{p}$ and takes $\mathcal{O}(n)$ time.
2. Line 2 (While Loop): The loop runs until J is empty, so the number of iterations depends on the number of elements in J. Clearly, it executes at most n times. Each iteration involves Lines 3–6, so we analyze each of these lines within the context of a single iteration.

   (a) Line 3: Select the last column k from J. In other words, we have to select the largest index k of J with $J[k] = 1$. Since we have implemented a stack, pushing each index j with $J[j] = 1$ into the stack from smallest one to the largest one, deleting J, and renaming the stack with J and deleting . Therefore, selecting the last column of J is equivalent to popping an element from J. Hence, it takes $\mathcal{O}(1)$ time.
   (b) Line 4: Set x_k based on $\hat{p}_{\alpha(k)}$. Retrieving the value from $\alpha(k)$ and assigning x_k takes $\mathcal{O}(1)$ time.
   (c) Line 5: Update $\hat{p}_i$ for all i such that $m_{ik} = 1$. Since $m_{ik} = 1$, this operation iterates over each index i of vertices $v_i \in N_G[v_k]$. Using the adjacency list, this step takes $\mathcal{O}(\deg(v_k) + 1)$ for each iteration.
   (d) Line 6: Remove k from J. J is implemented as a stack. Removing the last element takes $\mathcal{O}(1)$ time.

 doi:10.20944/preprints202411.2117.v1

The overall time complexity of the algorithm is:

$$\mathcal{O}\left(n + \sum_{k \in J}(deg_G(v_k) + 4)\right) = \mathcal{O}(n + m),$$

where n is the number of vertices and m is the number of edges in the graph. This linear time complexity makes the algorithm efficient for sparse graphs.

Following the discussion above, Algorithms 3 and 4 both run in $\mathcal{O}(n + m)$ time. Consequently, the theorem holds. □

4. Total $\{k\}$-Domination in Weighted Chordal Bipartite Graphs

The total $\{k\}$-domination problem in weighted chordal bipartite graphs can be efficiently solved by structural properties of these graphs. Specifically, we demonstrate how the neighborhood matrix of a chordal bipartite graph ordered by its weak elimination ordering forms a greedy matrix in standard greedy form. The following lemma establishes the foundational connection between weak elimination orderings and greedy matrices.

Lemma 4. *If the vertices of a chordal bipartite graph are ordered by its weak elimination ordering, then the neighborhood matrix of the graph is a greedy matrix in standard greedy form.*

Proof. Let G be a chordal bipartite graph with the vertices ordered by its weak elimination ordering $v_1, v_2, \ldots, v_n$, and let $\mathbf{N} = (n_{ij}) \in \mathbb{R}^{n \times n}$ be the neighborhood matrix of G. Since the neighborhood matrix of a chordal bipartite graph is totally balanced [49], $\mathbf{N}$ is totally balanced. We now check if $\mathbf{N}$ contains the Γ-matrix as a submatrix:

$$\Gamma = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

Assume that $\mathbf{N}$ contains the Γ-matrix as a submatrix, shown below with entries chosen by rows i, j and columns k, ℓ, where $i < j$ and $k < \ell$:

	k	ℓ
i	1	1
j	1	0

.

Since $\mathbf{N}$ is a neighborhood matrix, each a_{ii} is 0 for $1 \leq i \leq j$. It imples that $i \neq k$, $i \neq \ell$, and $j \neq k$. Furthermore, as bipartite graphs contain only cycles with even number of vertices, if $j = \ell$, then v_i, v_j, and v_k would form a cycle of odd number of vertices, which is impossible. Thus, $j \neq \ell$.

Consider three cases:

- **Case 1:** $i < k$. Then, v_i, v_j, v_k, and v_ℓ are all vertices of G_i. Since v_i is adjacent to both v_k and v_ℓ, the neighborhood relationship $N_{G_i}(v_k) \subseteq N_{G_i}(v_\ell)$ must hold. As $v_j \in N_{G_i}(v_k)$, it follows that $v_j \in N_{G_i}(v_\ell)$. However, this implies that v_j is adjacent to v_ℓ, which contradicts the assumption that v_j and v_ℓ are not adjacent.
- **Case 2:** $k < i < \ell$. In this case, v_i, v_j, v_k, and v_ℓ are all vertices of G_k. Since v_i is adjacent to both v_k and v_ℓ, the neighborhood relationship $N_{G_k}(v_i) \subseteq N_{G_k}(v_\ell)$ must hold. Consequently, $v_j \in N_{G_k}(v_i)$ implies $v_j \in N_{G_k}(v_\ell)$. However, this contradicts the assumption that v_j and v_ℓ are not adjacent.
- **Case 3:** $\ell < i$. Here, v_i, v_j, v_k, and v_ℓ are again vertices of G_k. By reasoning similar to Case 2, the adjacency of v_i to both v_k and v_ℓ implies $N_{G_k}(v_i) \subseteq N_{G_k}(v_\ell)$. Since $v_j \in N_{G_k}(v_i)$, we deduce that $v_j \in N_{G_k}(v_\ell)$, which means v_j and v_ℓ are adjacent. This contradicts the assumption that v_j and v_ℓ are not adjacent.

 doi:10.20944/preprints202411.2117.v1

These cases confirm that $\mathbf{N}$ is totally balanced and excludes the Γ-matrix as a submatrix. By Claim 1, we conclude that the neighborhood matrix of a chordal bipartite graph ordered by its weak elimination ordering is a greedy matrix in standard greedy form. □

The following lemma is similar to Lemma 1. It allows us to concentrate exclusively on non-negative vertex weights ($\mathbb{R}_{\geq 0}$).

Lemma 5. *Let w be a vertex-weight function of a graph $G = (V, E)$, and let $\gamma_{tk}(G, w)$ represent the the minimum labeling weight of a total $\{k\}$-dominating function for (G, w). Define $S = \{v \mid v \in V, w(v) < 0\}$ and let w' be a vertex-weight function such that $w'(v) = \max\{w(v), 0\}$ for each $v \in V$. Then, $\gamma_{tk}(G, w) = \gamma_{tk}(G, w') + k\sum_{v \in S} w(v)$.*

Proof. This lemma can be proved by the arguments similar to those for proving Lemma 1. □

Let G be a chordal bipartite graph with the vertices arranged in weak elimination ordering with $\mathbf{N} = (n_{ij}) \in \mathbb{R}^{n \times n}$ as its neighborhood matrix. Let $\mathbf{w} = (w_j) \in \mathbb{R}^n_{\geq 0}$, $\mathbf{x} = (x_j) \in \mathbb{R}^n$, and $\mathbf{p} = (p_i) \in \mathbb{R}^n$, where $p_i = k$ for $1 \leq i \leq n$.

The total $\{k\}$-domination problem for weighted chordal bipartite graphs can be formulated as the following integer linear program $IP_{tk}(G, w)$:

$$\begin{aligned} &IP_{tk}(G, w): \\ &\quad \text{minimize} \quad \mathbf{w}^T\mathbf{x} \\ &\quad \text{subject to} \quad \mathbf{N}\mathbf{x} \geq \mathbf{p} \\ &\qquad\qquad\quad \mathbf{x} \geq 0, \mathbf{x} \in \mathbb{Z}. \end{aligned}$$

Let $LP_{tk}(G, w)$ be the linear relaxation of $IP_{tk}(G, w)$, and let $LP^d_{tk}(G, w)$ be the dual program of $LP_{tk}(G, w)$. We present the formulations of P, D, $LP_{tk}(G, w)$, and $LP^d_{tk}(G, w)$ together below:

$$\begin{aligned} &\text{Primal } (P): && &&\text{Dual } (D): \\ &\quad \text{minimize} \quad \mathbf{c}^T\mathbf{x} + \mathbf{d}^T\mathbf{z} && &&\text{maximize} \quad \mathbf{b}^T\mathbf{y} \\ &\quad \text{subject to} \quad \mathbf{A}\mathbf{x} + \mathbf{z} \geq \mathbf{b} && &&\text{subject to} \quad \mathbf{A}^T\mathbf{y} \leq \mathbf{c} \\ &\qquad\qquad\quad \mathbf{x} \geq 0, \quad \mathbf{z} \geq 0. && &&\qquad\qquad\quad \mathbf{y} \geq 0, \quad \mathbf{y} \leq \mathbf{d}. \end{aligned} \tag{1}$$

$$\begin{aligned} &\text{Primal } (LP_{tk}(G, w)): && &&\text{Dual } (LP^d_{tk}(G, w)): \\ &\quad \text{minimize} \quad \mathbf{w}^T\mathbf{x} && &&\text{maximize} \quad \mathbf{p}^T\mathbf{y} \\ &\quad \text{subject to} \quad \mathbf{N}\mathbf{x} \geq \mathbf{p} && &&\text{subject to} \quad \mathbf{N}^T\mathbf{y} \leq \mathbf{w} \\ &\qquad\qquad\quad \mathbf{x} \geq 0. && &&\qquad\qquad\quad \mathbf{y} \geq 0. \end{aligned}$$

Theorem 7. *For weighted chordal bipartite graphs (G, w) with n vertices arranged by its weak elimination ordering, the total $\{k\}$-domination problem can be solved in $\mathcal{O}(n + m)$ time, where m is the number of edges in G.*

Proof. The proof proceeds in three steps:

1. **Neighborhood Matrix Properties**: By Lemma 4, we have established that for a weighted chordal bipartite graph (G, w) with n vertices arranged by its weak elimination ordering, the neighborhood matrix $\mathbf{N}$ of G is a greedy matrix in standard greedy form. This property ensures that the constraints of the total $\{k\}$-domination problem are well-structured for efficient computation.

 doi:10.20944/preprints202411.2117.v1

2. **Connection to Linear Programs**: The primal and dual linear programs, $LP_{tk}(G, w)$ and $LP^d_{tk}(G, w)$, for the weighted chordal bipartite graphs are specific instances of the linear programs P and D, where $\mathbf{z} = 0$ and the vector $\mathbf{d}$ is absent. These simplifications reduce the computational overhead associated with solving the general cases.
3. **Efficient Computation**: Using arguments similar to those in the proofs of Theorems 5 and 6, we use the greedy matrix property of $\mathbf{N}$ and advanced primal-dual algorithms to solve the relaxed problem efficiently. Optimized data structures ensure that each step in the algorithm operates in linear time relative to the size of the graph.

Combining these observations, the total $\{k\}$-domination problem for weighted chordal bipartite graphs (G, w) can be solved in $\mathcal{O}(n + m)$ time. □

5. Total k-Tuple Domination and k-Tuple Domination

The k-tuple domination and total k-tuple domination problems are fundamental in graph theory, with applications in optimization and network analysis. In this section, we investigate these problems for weighted proper interval graphs and weighted convex bipartite graphs. Utilizing structural properties of these graph classes, we formulate integer linear programs with totally unimodular constraint matrices, enabling efficient solutions via linear relaxations.

5.1. Integer Linear Programs with Totally Unimodular Matrices

Totally unimodular matrices allow integer solutions to be obtained directly from linear relaxations, making them crucial in solving the k-tuple and total k-tuple domination problems efficiently. This section presents our results about totally unimodular matrices and their applications.

5.1.1. Problem Formulations

Let (G, w) be a weighed graph with n vertices and m edges. The k-tuple domination problem for (G, w) is formulated as an integer linear program $IP_{\times k}(G, w)$:

$$\text{minimize } \mathbf{w}^T\mathbf{x}, \quad \text{subject to } \mathbf{M}\mathbf{x} \geq \mathbf{p},\ \mathbf{x} \in \{0,1\}^n,$$

where $\mathbf{M}$ is the closed neighborhood matrix of G, $\mathbf{w}$ is the weight vector, and $\mathbf{p}$ is a k-vector. Its linear relaxations $LP_{\times k}(G, w))$ is

$$\text{minimize } \mathbf{w}^T\mathbf{x}, \quad \text{subject to } \mathbf{M}\mathbf{x} \geq \mathbf{p},\ 0 \leq \mathbf{x} \leq 1.$$

To incorporate the constraint $\mathbf{x} \leq 1$ into the standard inequality framework, we express it equivalently as:

$$\mathbf{x} \leq 1 \quad \Longleftrightarrow \quad -\mathbf{x} \geq -1.$$

This transformation allows us to maintain a unified form for all constraints. Using this approach, the linear program $LP_{\times k}(G, w)$ can be rewritten as:

$$\text{minimize } \mathbf{w}^T\mathbf{x}, \quad \text{subject to } \mathbf{M}'\mathbf{x} \geq \mathbf{p}',\ \mathbf{x} \geq 0,$$

where $\mathbf{M}' = \begin{pmatrix} \mathbf{M} \\ -\mathbf{I} \end{pmatrix}$, $\mathbf{p}' = \begin{pmatrix} \mathbf{p} \\ -\mathbf{q} \end{pmatrix}$, $\mathbf{I}$ is the identity matrix, and $\mathbf{q}$ is a 1-vector.

Similarly, the total k-tuple domination problem is formulated as an integer linear program $IP_{\times tk}(G, w)$:

$$\text{minimize } \mathbf{w}^T\mathbf{x}, \quad \text{subject to } \mathbf{N}\mathbf{x} \geq \mathbf{p},\ \mathbf{x} \in \{0,1\}^n,$$

where $\mathbf{N}$ is the neighborhood matrix. Its linear relaxation $LP_{\times tk}(G, w))$ can be formulated as:

$$\text{minimize } \mathbf{w}^T\mathbf{x}, \quad \text{subject to } \mathbf{N}'\mathbf{x} \geq \mathbf{p}',\ \mathbf{x} \geq 0,$$

 doi:10.20944/preprints202411.2117.v1

where $\mathbf{N}' = \begin{pmatrix} \mathbf{N} \\ -\mathbf{I} \end{pmatrix}$, $\mathbf{p}' = \begin{pmatrix} \mathbf{p} \\ -\mathbf{q} \end{pmatrix}$, $\mathbf{I}$ is the identity matrix, and $\mathbf{q}$ is a 1-vector.

5.1.2. Fundamental Lemmas on Totally Unimodular Matrices

In this section, we present our fundamental lemmas on totally unimodular matrices.

Lemma 6. *Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ be a totally unimodular matrix, and let $\mathbf{I} \in \mathbb{R}^{n \times n}$ be the identity matrix. Then, the matrix*

$$\mathbf{B} = \begin{pmatrix} \mathbf{A} \\ -\mathbf{I} \end{pmatrix}$$

is also totally unimodular.

Proof. It is clear that the matrix $-\mathbf{I}$ is totally unimodular, as each determinant of any square submatrix of $-\mathbf{I}$ evaluates to $-1, 0$, or 1, satisfying the definition of total unimodularity. Let $\mathbf{B}' = (b'_{ij}) \in \mathbb{R}^{p \times p}$ be any square submatrix of $\mathbf{B}$.

First, if the rows of $\mathbf{B}'$ are formed exclusively from either $\mathbf{A}$ or $-\mathbf{I}$, then $\mathbf{B}'$ is totally unimodular because $\mathbf{A}$ and $-\mathbf{I}$ satisfy the definition of total unimodularity by ensuring that every determinant of their square submatrices is in $\{-1, 0, 1\}$.

Next, consider the case where the rows of $\mathbf{B}'$ are formed from both $\mathbf{A}$ and $-\mathbf{I}$. We compute $\det(\mathbf{B}')$ using the Laplace expansion along a row i formed from $-\mathbf{I}$. By the definition of Laplace expansion:

$$\det(\mathbf{B}') = \sum_{j=1}^{p} b'_{ij} (-1)^{i+j} \det(\mathbf{B}'_{ij}),$$

where $\mathbf{B}'_{ij}$ is the $(p-1) \times (p-1)$ square submatrix obtained by removing the i-th row and j-th column from $\mathbf{B}'$.

Since row i comes from $-\mathbf{I}$, it has at most one nonzero entry, which is -1. This means that:

$$\det(\mathbf{B}') = 0 \quad \text{or} \quad \det(\mathbf{B}') = (-1)^s \det(\mathbf{B}'_{ij}),$$

where s is a non-negative integer determined by the row and column indices of the nonzero entry.

If $\mathbf{B}'_{ij}$ still contains rows from $-\mathbf{I}$, we recursively apply the Laplace expansion and remove rows and columns associated with $-\mathbf{I}$. Each step multiplies the determinant by $(-1)^s$, with $s \in \{0, 1\}$. Eventually, the process reduces $\mathbf{B}'$ to a square submatrix of $\mathbf{A}$.

Since $\mathbf{A}$ satisfies the definition of total unimodularity, every determinant of its square submatrices is in $\{-1, 0, 1\}$. Therefore, $\det(\mathbf{B}') \in \{-1, 0, 1\}$ as well.

Thus, $\mathbf{B}$ is totally unimodular. □

Lemma 7. *Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ be a totally unimodular matrix, and let $\mathbf{I} \in \mathbb{R}^{m \times m}$ be the identity matrix. Then, the matrix*

$$\begin{pmatrix} \mathbf{A} & -\mathbf{I} \end{pmatrix}$$

is also totally unimodular.

Proof. By Lemma 6, we know that the matrix

$$\mathbf{B} = \begin{pmatrix} \mathbf{A} \\ -\mathbf{I} \end{pmatrix}$$

 doi:10.20944/preprints202411.2117.v1

is totally unimodular. Take the transpose of $\mathbf{B}$ to obtain $\begin{pmatrix}\mathbf{A} & -\mathbf{I}\end{pmatrix}$. The determinant of each square submatrix remains unchanged by transposing, so $\det(\mathbf{M}) = \det(\mathbf{M}^T)$ for each square submatrix $\mathbf{M}$ of $\begin{pmatrix}\mathbf{A} & -\mathbf{I}\end{pmatrix}$. Consequently, $\begin{pmatrix}\mathbf{A} & -\mathbf{I}\end{pmatrix}$ is a totally unimodular matrix. □

Theorem 8 ([45]). *Let* $\mathbf{A}$ *be an integral matrix. Then,* $\mathbf{A}$ *is totally unimodular if and only if for each integral vector* $\mathbf{b}$*, the polyhedron*

$$\{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$$

is integral.

Lemma 8. *Let* $\mathbf{A}$ *be an integral matrix. Then,* $\mathbf{A}$ *is totally unimodular if and only if for each integral vector* $\mathbf{b}$*, the polyhedron*

$$\{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} \geq \mathbf{b}\}$$

is integral.

Proof. Assume that $\mathbf{A}$ contains r rows. By the properties of determinants under scalar multiplication ([47]), for each square submatrix $\mathbf{A}'$ of $\mathbf{A}$, we have

$$\det(-\mathbf{A}') = (-1)^r \cdot \det(\mathbf{A}').$$

If $\mathbf{A}$ is totally unimodular, then $\det(-\mathbf{A}') \in \{-1, 0, 1\}$, which implies that $-\mathbf{A}$ is totally unimodular. Let $\mathbf{s}$ be an integer vector. Clearly,

$$\mathbf{s} \in \{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} \geq \mathbf{b}\} \text{ if and only if } \mathbf{s} \in \{\mathbf{x} \mid \mathbf{x} \geq 0, -\mathbf{A}\mathbf{x} \leq -\mathbf{b}\}.$$

By Theorem 8, the polyhedron $\{\mathbf{x} \mid \mathbf{x} \geq 0, -\mathbf{A}\mathbf{x} \leq -\mathbf{b}\}$ is integral. Therefore, $\{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} \geq \mathbf{b}\}$ is also integral.

Conversely, if $\{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} \geq \mathbf{b}\}$ is integral, then so is $\{\mathbf{x} \mid \mathbf{x} \geq 0, -\mathbf{A}\mathbf{x} \leq -\mathbf{b}\}$. By Theorem 8, $-\mathbf{A}$ is totally unimodular. Following the same reasoning as above, $-(-\mathbf{A}) = \mathbf{A}$ is therefore also totally unimodular.

Thus, the lemma holds in both directions. □

Lemma 9. *Let* $\mathbf{A}$ *be a totally unimodular matrix. For each integral vector* $\mathbf{b}$*, the polyhedron*

$$\{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} = \mathbf{b}\}$$

is integral.

Proof. It is clear that the polyhedron $\{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} = \mathbf{b}\}$ is equivalent to the following:

$$\{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{A}\mathbf{x} \leq \mathbf{b}\} = \{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} \geq \mathbf{b}\} \cap \{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} \leq \mathbf{b}\}.$$

Following Theorem 8 and Lemma 8, the polyhedra $\{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} \geq \mathbf{b}\}$ and $\{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ are integral. Therefore, $\{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} = \mathbf{b}\}$ is integral. □

5.2. k-Tuple Domination in Weighted Proper Interval Graphs

We have formulated the k-tuple domination problem for weighted graphs (G, w) as the integer linear program $IP_{\times k}(G, w)$. For weighted proper interval graphs, solving $IP_{\times k}(G, w)$ is equivalent to directly solving the k-tuple domination problem.

The following lemma establishes a critical connection between the closed neighborhood matrices of weighted proper interval graphs and totally unimodular matrices. This connection enables $IP_{\times k}(G, w)$ to be solved efficiently by addressing its linear relaxation $LP_{\times k}(G, w)$.

 doi:10.20944/preprints202411.2117.v1

Lemma 10. *The closed neighborhood matrix of a weighted proper interval graph is totally unimodular.*

Proof. The lemma follows from two established results:

- A graph is a proper interval graph if and only if its closed neighborhood matrix has the consecutive ones property for columns [18].
- If a $(0,1)$-matrix has the consecutive ones property for columns, then it is totally unimodular [45].

□

Theorem 9. *The k-tuple domination problem for weighted proper interval graphs (G,w) can be solved in the running time of*

$$\mathcal{O}\Big(n^{2.371552}\log^2(n)\log(n/\delta)\Big),$$

where n is the number of vertices, and δ is the desired accuracy within the range $(0,1]$.

Proof. Following Lemmas 6, 8, and 10, we know that the closed neighborhood matrix $\mathbf{M}$ of G and the constraint matrix $\mathbf{M}'$ in $LP_{\times k}(G,w)$ are totally unimodular, and the polyhedron of $LP_{\times k}(G,w)$ is integer. Therefore, the optimal objective values of $LP_{\times k}(G,w)$ and $IP_{\times k}(G,w)$ are equal. Thus, solving $\mathrm{IP}_{\times tk}(G,w)$ is equivalent to finding an integral solution to $\mathrm{LP}_{\times tk}(G,w)$. This implies that the total k-tuple domination problem for weighted convex bipartite graphs can be solved by computing an integral solution to $\mathrm{LP}_{\times tk}(G,w)$.

Recently, van den Brand [55] derandomized the algorithm Cohen et al. [56] to achieve that linear programs of the form $\min\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \geq 0,\ \mathbf{A}\mathbf{x} = \mathbf{b}\}$ with no redundant constraints can be solved in the running time

$$\mathcal{O}\Big((n^{\omega} + n^{2.5-\alpha/2+\wr(1)} + n^{2+1/6+\wr(1)})\log^2(n)\log(n/\delta)\Big),$$

where n is the number of decision variables, ω is the exponent of matrix multiplication, α is the dual exponent of matrix multiplication, and δ is the desired accuracy within the range $(0,1]$. Williams et al. [57] established $\omega \leq 2.371552$ and $\alpha \geq 0.321334$. This yields a simplified complexity of $\mathcal{O}\Big(n^{2.371552}\log^2(n)\log(n/\delta)\Big)$.

The linear program $LP_{\times k}(G,w)$ is $\min\{\mathbf{w}^T\mathbf{x} \mid \mathbf{x} \geq 0,\ \mathbf{M}'\mathbf{x} \geq \mathbf{p}'\}$. To solve $LP_{\times k}(G,w)$ by van den Brand's algorithm, we transform $LP_{\times k}(G,w)$ into the form $\min\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \geq 0,\ \mathbf{A}\mathbf{x} = \mathbf{b}\}$ using *slack variables*. The transformation steps are as follows:

1. **Introduce Slack Variables.**
   Convert the inequality $\mathbf{M}'\mathbf{x} \geq \mathbf{p}'$ into an equality by introducing slack variables. Rewrite it as:

   $$\mathbf{M}'\mathbf{x} = \mathbf{p}' + \mathbf{s},$$

   where $\mathbf{s} \geq 0$ is a vector of slack variables. This ensures each constraint in the original inequality has a corresponding non-negative slack variable.
2. **Rewrite as Equalities.**
   Rearrange the equality above as:

   $$\mathbf{M}'\mathbf{x} - \mathbf{s} = \mathbf{p}'.$$

   Define a new vector $\tilde{\mathbf{x}} = \begin{pmatrix}\mathbf{x}\\ \mathbf{s}\end{pmatrix}$, so we can express the constraints in matrix form as:

   $$\begin{pmatrix}\mathbf{M}' & -\mathbf{I}\end{pmatrix}\tilde{\mathbf{x}} = \mathbf{p}'.$$

 doi:10.20944/preprints202411.2117.v1

3. **Extend the Objective Function.**
   Since the original objective function is $\mathbf{w}^T\mathbf{x}$, extend it to include the slack variables by setting:

$$\mathbf{c} = \begin{pmatrix} \mathbf{w} \\ \mathbf{j} \end{pmatrix},$$

   where $\mathbf{j}$ is a zero vector. This ensures that the slack variables $\mathbf{s}$ do not affect the objective function.

   Thus, the transformed linear program becomes:

$$\min\{\mathbf{c}^T\tilde{\mathbf{x}} \mid \tilde{\mathbf{x}} \geq 0,\ \mathbf{A}\tilde{\mathbf{x}} = \mathbf{b}\},$$

where $\mathbf{A} = \begin{pmatrix} \mathbf{M}' & -\mathbf{I} \end{pmatrix}$, $\mathbf{b} = \mathbf{p}'$, $\tilde{\mathbf{x}} = \begin{pmatrix} \mathbf{x} \\ \mathbf{s} \end{pmatrix}$, and $\mathbf{c} = \begin{pmatrix} \mathbf{w} \\ \mathbf{j} \end{pmatrix}$.

We verify the correctness and analyze the time complexity of this transformation by confirming that the feasible sets of $\mathbf{x}$ and objective functions of the original and transformed problems are equivalent. The steps are as follows:

1. **Converting Inequalities to Equalities via Slack Variables**
   In $LP_{timesk}(G, w)$, the inequality $\mathbf{M}'\mathbf{x} \geq \mathbf{p}'$ restricts $\mathbf{x}$ so that each entry of $\mathbf{M}'\mathbf{x}$ is at least the corresponding entry of $\mathbf{p}'$. To convert this inequality into an equality, we introduce a vector of slack variables $\mathbf{s} \geq 0$, rewriting it as:

$$\mathbf{M}'\mathbf{x} = \mathbf{p}' + \mathbf{s}.$$

   This ensures that $\mathbf{M}'\mathbf{x} \geq \mathbf{p}'$ holds if and only if there exists a non-negative vector $\mathbf{s}$ such that $\mathbf{M}'\mathbf{x} = \mathbf{p}' + \mathbf{s}$. Since $\mathbf{M}'$ has $2n$ rows and n columns, the vector $\mathbf{s}$ consists of $2n$ entries. Setting up the slack variables takes $\mathcal{O}(n)$ time.
2. **Reformulating Constraints with Augmented Matrices**
   We define $\tilde{\mathbf{x}} = \begin{pmatrix} \mathbf{x} \\ \mathbf{s} \end{pmatrix}$ and rewrite the equality as:

$$\mathbf{A}\tilde{\mathbf{x}} = \mathbf{b},$$

   where $\mathbf{A} = \begin{pmatrix} \mathbf{M}' & -\mathbf{I} \end{pmatrix}$ and $\mathbf{b} = \mathbf{p}'$. This matrix formulation ensures that each slack variable s_i directly accounts for the surplus required to satisfy each inequality. Constructing $\mathbf{A}$ requires combining $\mathbf{M}'$ with $-\mathbf{I}$, and constructing $\tilde{\mathbf{x}}$ requires combining $\mathbf{x}$ with $\mathbf{s}$.

   The matrix $\mathbf{A}$ has $2n$ rows and $3n$ columns. The vector $\tilde{\mathbf{x}}$ has $3n$ rows and one column, resulting in an augmented matrix, which takes $\mathcal{O}(n^2)$ time. Hence, the step can be done in $\mathcal{O}(n^2)$ time.
3. **Ensuring Equivalence of Feasible Sets of x**
   In the original linear program, the feasible region is defined by all $\mathbf{x} \geq 0$ such that $\mathbf{M}'\mathbf{x} \geq \mathbf{p}'$. In the transformed linear program, the feasible region is defined by all non-negative vectors $\tilde{\mathbf{x}} = \begin{pmatrix} \mathbf{x} \\ \mathbf{s} \end{pmatrix} \geq 0$ that satisfy $\mathbf{A}\tilde{\mathbf{x}} = \mathbf{b}$. Clearly,

$$\mathbf{x} \in \{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{M}'\mathbf{x} \geq \mathbf{p}'\} \text{ if and only if } \mathbf{x} \in \{\mathbf{x} \mid \mathbf{x} \geq 0, \mathbf{s} \geq 0, \mathbf{M}'\mathbf{x} = \mathbf{p}' + \mathbf{s}\}.$$

   This establishes that the transformation preserves the feasible region.
4. **Ensuring Equivalence of Objective Functions**
   The original objective function $\mathbf{w}^T\mathbf{x}$ depends only on $\mathbf{x}$, so in the transformed program, we extend it by defining $\mathbf{c} = \begin{pmatrix} \mathbf{w} \\ \mathbf{j} \end{pmatrix}$, where $\mathbf{j}$ is a zero vector. This construction prevents the slack variables $\mathbf{s}$ from influencing the objective function, so that the objective $\mathbf{c}^T\tilde{\mathbf{x}} = \mathbf{w}^T\mathbf{x}$ in the transformed program is identical to the original objective. Extending the objective vector takes $\mathcal{O}(n)$ time.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 27 November 2024 doi:10.20944/preprints202411.2117.v1

Having verified that the transformation preserves both the feasible region and the objective function, we conclude that the transformation is correct with the time complexity of $\mathcal{O}(n^2)$.

Since $\mathbf{M}'$ is totally unimodular, Lemmas 7 and 9 imply that $\mathbf{A} = \begin{pmatrix} \mathbf{M}' & -\mathbf{I} \end{pmatrix}$ is totally unimodular, and the polyhedron of the transformed linear program is integer.

Despite the transformed linear program containing $3n$ variables, it can still be solved in $\mathcal{O}\left(n^{2.371552} \log^2(n) \log(n/\delta)\right)$ time. Consequently, the linear relaxation $LP_{\times k}(G, w)$ runs in time of:

$$\mathcal{O}\left(n^2 + n^{2.371552} \log^2(n) \log(n/\delta)\right) = \mathcal{O}\left(n^{2.371552} \log^2(n) \log(n/\delta)\right).$$

□

5.3. Total k-Tuple Domination in Weighted Convex Bipartite Graphs

In this subsection, we explore the total k-tuple domination problem in weighted convex bipartite graphs. In Section 5.1.1, we formulated the total k-tuple domination problem for weighted graphs (G, w) as the integer linear program $IP_{\times tk}(G, w)$. For weighted convex bipartite graphs (G, w), solving $IP_{\times tk}(G, w)$ is equivalent to directly solving the total k-tuple domination problem.

To establish a connection between the neighborhood matrices of weighted convex bipartite graphs and totally unimodular matrices (as discussed in Sections 5.1 and 5.2), we begin by examining the neighborhood matrix of a convex bipartite graph.

Figure 9 illustrates a convex bipartite graph $G = (X, Y, E)$, where $X = \{x_1, x_2, x_3, x_4\}$ and $Y = \{y_1, y_2, y_3, y_4\}$.

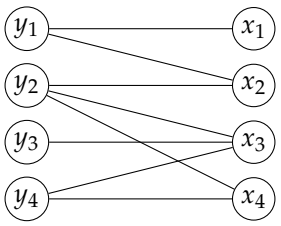


Figure 9. A convex bipartite graph $G = (X, Y, E)$, where $X = \{x_1, x_2, x_3, x_4\}$ and $Y = \{y_1, y_2, y_3, y_4\}$. The graph satisfies the adjacency property for the ordering of X.

The neighborhood matrix $\mathbf{N}$ of G is shown below:

	x_1	x_2	x_3	x_4	y_1	y_2	y_3	y_4
x_1	0	0	0	0	1	0	0	0
x_2	0	0	0	0	1	1	0	0
x_3	0	0	0	0	0	1	1	1
x_4	0	0	0	0	0	1	0	1
y_1	1	1	0	0	0	0	0	0
y_2	0	1	1	1	0	0	0	0
y_3	0	0	1	0	0	0	0	0
y_4	0	0	1	1	0	0	0	0

$$= \begin{pmatrix} \mathbf{A}_1 & \mathbf{A}_2 \\ \mathbf{A}_3 & \mathbf{A}_4 \end{pmatrix}$$

The matrix $\mathbf{N}$ is partitioned into four blocks: $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, and $\mathbf{A}_4$. Since G is bipartite, no two vertices in X or Y are adjacent. Therefore, $\mathbf{A}_1$ and $\mathbf{A}_4$ are zero-matrices. Moreover, the adjacency property ensures that every column of $\mathbf{A}_2$ has consecutive ones. Then, $\mathbf{A}_2$ has the consecutive ones property for columns. The block $\mathbf{A}_3$ is the transpose of $\mathbf{A}_2$ because of the symmetry of a neighborhood matrix of a graph.

 doi:10.20944/preprints202411.2117.v1

Lemma 11. *Let* $\mathbf{M}$ *be a square matrix partitioned as*

$$\mathbf{M} = \begin{pmatrix} \mathbf{Q} & \mathbf{P} \\ \mathbf{S} & \mathbf{R} \end{pmatrix},$$

where $\mathbf{Q}$ *is a zero-matrix, and* $\mathbf{P}$ *and* $\mathbf{S}$ *are square matrices with* $\det(\mathbf{P}), \det(\mathbf{S}) \in \{-1, 0, 1\}$. *Then,*

$$\det(\mathbf{M}) \in \{-1, 0, 1\}.$$

Proof. By a factorization involving the Schur complement [58], we have

$$\det\begin{pmatrix} \mathbf{P} & \mathbf{Q} \\ \mathbf{R} & \mathbf{S} \end{pmatrix} = \det(\mathbf{P}) \det(\mathbf{S}),$$

where $\mathbf{P}$ and $\mathbf{S}$ contribute independently to the determinant. Since $\det(\mathbf{P}), \det(\mathbf{S}) \in \{-1, 0, 1\}$, it follows that

$$\det\begin{pmatrix} \mathbf{P} & \mathbf{Q} \\ \mathbf{R} & \mathbf{S} \end{pmatrix} \in \{-1, 0, 1\}.$$

Next, let

$$\mathbf{M}' = \begin{pmatrix} \mathbf{P} & \mathbf{Q} \\ \mathbf{R} & \mathbf{S} \end{pmatrix}.$$

To transform $\mathbf{M}'$ into $\mathbf{M}$, we perform column exchanges. Each column exchange introduces a sign change in the determinant. Therefore, we have

$$\det(\mathbf{M}) = (-1)^s \det(\mathbf{M}'),$$

where s is the number of column exchanges.

Since $\det(\mathbf{M}') \in \{-1, 0, 1\}$, and sign changes do not affect the magnitude of the determinant, we conclude

$$\det(\mathbf{M}) \in \{-1, 0, 1\}.$$

□

Theorem 10. *Let* $G = (X, Y, E)$ *be a convex bipartite graph, where the ordering of* X *satisfies the adjacency property. Then, the neighborhood matrix* $\mathbf{N}$ *of* G *is totally unimodular.*

Proof. Let $X = \{1, 2, \ldots, r\}$ and $Y = \{r+1, r+2, \ldots, n\}$. The neighborhood matrix $\mathbf{N}$ of G, an $n \times n$ matrix, can be partitioned as

$$\mathbf{N} = \begin{pmatrix} \mathbf{N}_1 & \mathbf{N}_2 \\ \mathbf{N}_3 & \mathbf{N}_4 \end{pmatrix},$$

where $\mathbf{N}_1 = \mathbf{N}(1:r, 1:r)$, $\mathbf{N}_2 = \mathbf{N}(1:r, r+1:n)$, $\mathbf{N}_3 = \mathbf{N}(r+1:n, 1:r)$, $\mathbf{N}_4 = \mathbf{N}(r+1:n, r+1:n)$. Since G is bipartite, $\mathbf{N}_1$ and $\mathbf{N}_4$ are zero-matrices.

The adjacency property of G ensures that $\mathbf{N}_2$ satisfies the consecutive ones property for columns. By Theorem 1, $\mathbf{N}_2$ is totally unimodular. Since $\mathbf{N}_3 = \mathbf{N}_2^T$ and the transpose of a totally unimodular matrix is also totally unimodular, $\mathbf{N}_3$ is totally unimodular.

To verify that $\mathbf{N}$ is totally unimodular, we show that every square submatrix $\mathbf{N}'$ of $\mathbf{N}$ satisfies $\det(\mathbf{N}') \in \{-1, 0, 1\}$. Assume that $\mathbf{N}'$ is composed of p rows and p columns, where $1 \leq p \leq n$. We consider three cases based on the structure of $\mathbf{N}'$.

 doi:10.20944/preprints202411.2117.v1

Case 1: $\mathbf{N}'$ Is a Submatrix of a Single Block.

If $\mathbf{N}'$ is a submatrix of $\mathbf{N}_1$ or $\mathbf{N}_4$, it is a zero-matrix, so $\det(\mathbf{N}') = 0$. If $\mathbf{N}'$ is a submatrix of $\mathbf{N}_2$ or $\mathbf{N}_3$, the total unimodularity of $\mathbf{N}_2$ and $\mathbf{N}_3$ ensures $\det(\mathbf{N}') \in \{-1, 0, 1\}$.

Case 2: $\mathbf{N}'$ Includes Rows and Columns from Exactly Two Blocks.

The rows and columns of $\mathbf{N}'$ are from two blocks, $\mathbf{N}_1$ and $\mathbf{N}_2$, $\mathbf{N}_2$ and $\mathbf{N}_4$, $\mathbf{N}_3$ and $\mathbf{N}_4$, or $\mathbf{N}_1$ and $\mathbf{N}_3$. Since $\mathbf{N}_1$ and $\mathbf{N}_4$ are zero-matrices, at least one row or column of $\mathbf{N}'$ is entirely zero, implying $\det(\mathbf{N}') = 0$.

Case 3: $\mathbf{N}'$ Includes Rows and Columns from All Four Blocks.

We partition $\mathbf{N}'$ as

$$\mathbf{N}' = \begin{pmatrix} \mathbf{N}'_1 & \mathbf{N}'_2 \\ \mathbf{N}'_3 & \mathbf{N}'_4 \end{pmatrix},$$

where $\mathbf{N}'_1 = \mathbf{N}'(1:s, 1:t)$, $\mathbf{N}'_2 = \mathbf{N}'(1:s, t+1:p)$, $\mathbf{N}'_3 = \mathbf{N}'(s+1:p, 1:t)$, $\mathbf{N}'_4 = \mathbf{N}'(s+1:p, t+1:p)$.

In this case, $\mathbf{N}'_i$ is a submatrix of $\mathbf{N}_i$ for $1 \leq i \leq 4$. Since $\mathbf{N}'_1$ and $\mathbf{N}'_4$ are submatrices of zero-matrices, they are zero-matrices themselves. This reduces $\mathbf{N}'$ to a dependency on $\mathbf{N}'_2$ and $\mathbf{N}'_3$. We analyze based on the dimensions of the submatrices.

Case 3.1: $p - t < s$.

In this case, $p - s < t$. We partition $\mathbf{N}'$ as

$$\mathbf{N}' = \begin{pmatrix} \mathbf{N}''_1 & \mathbf{N}''_2 \\ \mathbf{N}''_3 & \mathbf{N}''_4 \end{pmatrix},$$

where $\mathbf{N}''_1 = \mathbf{N}'(1:s, 1:p-s)$, $\mathbf{N}''_2 = \mathbf{N}'(1:s, p-s+1:p)$, $\mathbf{N}''_3 = \mathbf{N}'(s+1:p, 1:p-s)$, $\mathbf{N}''_4 = \mathbf{N}'(s+1:p, p-s+1:p)$.

Here, $\mathbf{N}''_1$ is a zero-matrix, and both $\mathbf{N}''_2$ and $\mathbf{N}''_3$ are square matrices. Notably, $\mathbf{N}''_2$ contains a column of zeros because $p - s < t$. We have $\det(\mathbf{N}''_2) = 0$. Since $\mathbf{N}''_3$ is a submatrix of $\mathbf{N}_3$, it preserves totally unimodularity, and thus $\det(\mathbf{N}''_3) \in \{-1, 0, 1\}$. By Lemma 11, $\det(\mathbf{N}') \in \{-1, 0, 1\}$.

Case 3.2: $p - t > s$.

In this case, $p - s > t$. We partition $\mathbf{N}'$ as

$$\mathbf{N}' = \begin{pmatrix} \mathbf{N}''_1 & \mathbf{N}''_2 \\ \mathbf{N}''_3 & \mathbf{N}''_4 \end{pmatrix},$$

where $\mathbf{N}''_1 = \mathbf{N}'(1:p-t, 1:t)$, $\mathbf{N}''_2 = \mathbf{N}'(1:p-t, t+1:p)$, $\mathbf{N}''_3 = \mathbf{N}'(p-t+1:p, 1:t)$, $\mathbf{N}''_4 = \mathbf{N}'(p-t+1:p, t+1:p)$.

Here, $\mathbf{N}''_1$ is a zero-matrix, and both $\mathbf{N}''_2$ and $\mathbf{N}''_3$ are square matrices. Notably, $\mathbf{N}''_2$ contains a row of zeros because $p - t > s$. We have $\det(\mathbf{N}''_2) = 0$. Since $\mathbf{N}''_3$ is a submatrix of $\mathbf{N}_3$, it preserves totally unimodularity, and thus $\det(\mathbf{N}''_3) \in \{-1, 0, 1\}$. By Lemma 11, $\det(\mathbf{N}') \in \{-1, 0, 1\}$.

Case 3.3: $p - t = s$.

In this case, $p - s = t$. We partition $\mathbf{N}'$ as

$$\mathbf{N}' = \begin{pmatrix} \mathbf{N}''_1 & \mathbf{N}''_2 \\ \mathbf{N}''_3 & \mathbf{N}''_4 \end{pmatrix},$$

 doi:10.20944/preprints202411.2117.v1

where $\mathbf{N}_1'' = \mathbf{N}'(1:s, 1:t)$, $\mathbf{N}_2'' = \mathbf{N}'(1:s, t+1:p)$, $\mathbf{N}_3'' = \mathbf{N}'(s+1:p, 1:t)$, $\mathbf{N}_4'' = \mathbf{N}'(s+1:p, t+1:p)$.

Here, $\mathbf{N}_1''$ is a zero-matrix, and both $\mathbf{N}_2''$ and $\mathbf{N}_3''$ are square matrices. Since $\mathbf{N}_2''$ and $\mathbf{N}_3''$ are submatrices of totally unimodular matrices, their determinants satisfy $\det(\mathbf{N}_2'') \in \{-1, 0, 1\}$ and $\det(\mathbf{N}_3'') \in \{-1, 0, 1\}$. By Lemma 11, $\det(\mathbf{N}') \in \{-1, 0, 1\}$.

In all subcases, $\det(\mathbf{N}') \in \{-1, 0, 1\}$. Therefore, the neighborhood matrix $\mathbf{N}$ of the convex bipartite graph G is totally unimodular. □

Theorem 11. *The total k-tuple domination problem for weighted convex bipartite graphs (G, w) can be solved in the running time of*

$$\mathcal{O}\left(n^{2.371552} \log^2(n) \log\left(\frac{n}{\delta}\right)\right),$$

where n is the number of vertices in G, and δ is the desired accuracy within the range $(0, 1]$.

Proof. By Lemma 6, Lemma 8, and Theorem 10, the neighborhood matrix $\mathbf{N}$ of G and the constraint matrix $\mathbf{N}'$ in $\mathrm{LP}_{\times tk}(G, w)$ are totally unimodular. Consequently, the polyhedron associated with $\mathrm{LP}_{\times tk}(G, w)$ is integer, meaning that the optimal objective values of $\mathrm{LP}_{\times tk}(G, w)$ and $\mathrm{IP}_{\times tk}(G, w)$ are equal.

Thus, solving $\mathrm{IP}_{\times tk}(G, w)$ is equivalent to finding an integral solution to $\mathrm{LP}_{\times tk}(G, w)$. This implies that the total k-tuple domination problem for weighted convex bipartite graphs can be solved by computing an integral solution to $\mathrm{LP}_{\times tk}(G, w)$.

Following arguments analogous to those used in the proof of Theorem 9, we conclude that the total k-tuple domination problem can be solved in the running time of

$$\mathcal{O}\left(n^{2.371552} \log^2(n) \log\left(\frac{n}{\delta}\right)\right).$$

□

6. Conclusions and Future Directions

This paper investigates the complexity and algorithmic solutions for $\{k\}$-domination, k-tuple domination, and their total domination variants in weighted subclasses of chordal graphs and bipartite graphs. The primary contributions of this work are as follows:

1. Developing efficient $\mathcal{O}(n + m)$ time algorithms for $\{k\}$-domination in weighted strongly chordal graphs and total $\{k\}$-domination in weighted chordal bipartite graphs.
2. Establishing the running time of $\mathcal{O}(n^{2.371552} \log^2(n) \log(n/\delta))$ for k-tuple and total k-tuple domination in weighted proper interval graphs and convex bipartite graphs. This result improves the running time for the k-tuple domination problem in unit proper interval graphs.
3. Extending theoretical models to vertex-weighted graph settings, bridging gaps in the existing research on weighted graph domination.
4. Leveraging linear and integer programming techniques, supported by totally balanced and totally unimodular matrices, to provide formal proofs of correctness and efficiency for the proposed algorithms.

These results advance both the theoretical understanding and computational efficiency of domination problems in weighted graph classes.

Future work can extend these domination techniques to other weighted graph classes, including real-world network applications and dynamic graph settings. Additionally, integrating these approaches with advanced optimization techniques, such as parallel computation and distributed systems, may yield even more efficient and scalable algorithms.

Funding: This research received no external funding.

Data Availability Statement: Data is contained within the article or supplementary material.

Acknowledgments: The author sincerely thanks the reviewers for their insightful comments and suggestions, which have greatly improved this paper's clarity, analysis, and overall quality. Their constructive feedback and dedication to advancing research in this field are deeply appreciated.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Hedetniemi, S.T.; Laskar, R.C. (Eds.), *Special Volume: Topics on Domination, Discrete Mathematics*, Vol. 86, No.1-3, December 1990.
2. Haynes, T.W.; Hedetniemi, S.T.; Slater, P.J. (Eds.), *Fundamentals of Domination in Graphs*, Marcel Dekker, New York, 1998.
3. Haynes, T.W.; Hedetniemi, S.T.; Slater, P.J. (Eds.), *Domination in Graphs: Advanced Topics*, Marcel Dekker, New York, 1998.
4. Haynes, T.W.; Hedetniemi, S.T.; Henning, M.A. (Eds.), *Topics in Domination in Graphs,Developments in Mathematics*, Vol. 64, Springer, Cham, 2020.
5. Haynes, T.W.; Hedetniemi, S.T.; Henning, M.A. (Eds.), *Structures of Domination in Graphs*, Developments in Mathematics, Vol. 66, Springer, Cham, 2021.
6. Haynes, T.W.; Hedetniemi, S.T.; Henning, M.A. (Eds.), *Domination in Graphs: Core Concepts*, Springer Monographs in Mathematics, Springer, Cham, 2023
7. Henning, M.A.; Yeo, A. *Total Domination in graphs*, Springer Monographs in Mathematics, Springer, 2013.
8. Argiroffo, G.; Leoni, V.; Torres, P. On the complexity of $\{k\}$-domination and k-tuple domination in graphs. *Inform. Process. Lett.* **2015**, *115*, 556-561.
9. Pradhan, D. Complexity of certain functional variants of total domination in chordal bipartite graphs. *Discrete Math. Algorithms Appl.* **2012**, *4*, Article 1240045.
10. Argiroffo, G.; Leoni, V.; Torres, P. Complexity of k-tuple total and total $\{k\}$-dominations for some subclasses of bipartite graphs. *Inform. Process. Lett.* **2018**, *138*, 75-80.
11. He, J.; Liang, H. Complexity of total $\{k\}$-domination and related problem. In Proceedings of the FAW-AAIM 2011, LNCS 6681, pp. 147-155.
12. Liao, C.; Chang, G.J. k-tuple domination in graphs. *Inform. Process. Lett.* **2003**, *87*, 45-50.
13. Li, P.; Wang, A.; Shang, J. A simple optimal algorithm for k-tuple dominating problem in interval graphs. *J. Comb. Optim.* **2023**, *45*, Article Number 14.
14. Dobson, M.P.; Leoni, V.; Lopez Pujato, M.I. Efficient algorithms for tuple domination on co-biconvex graphs and web graphs. arXiv preprint arXiv:2008.05345, 2022.
15. Lee, C-M.; Chang, M.S. Variations of Y-dominating functions on graphs. *Discrete Math.* **2008**, *308*, 4185-4204.
16. Chiarelli, N.; Hartinger, T.R.; Leoni, V.A.; Lopez Pujato, M.I.; Milaniĉ, M. New algorithms for weighted k-domination and total k-domination problem in proper interval graphs. *Theor. Comput. Sci.* **2019**, *795*, 128-141.
17. Li, P.; Li, X.; Liu, J.-B.; Shang, J. Optimized algorithms for problems related to weighted k-domination, k-tuple domination, and total k-domination for unit interval graphs, 2024. Available at SSRN: https://ssrn.com/abstract=4725957 or http://dx.doi.org/10.2139/ssrn.4725957.
18. Brandstädt, A.; Le, V. B.; Spinrad, J. P. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, 1999.
19. Hoffman, A.J.; Kolen, A.W.J.; Sakarovitch, M. Totally-balanced and greedy matrices. *Siam J. Alg. Disc. Meth.* **1985**, *6*, 721-730.
20. Argiroffo, G.; Leoni, V.; Torres, P. On the complexity of the labeled domination problem in graphs. *Int. Trans. Oper. Res.* **2017**, *24*, 355-367.
21. Bonomo-Braberman, F.; Gonzalez, C.L. A new approach on locally checkable problems. *Discrete Appl. Math.* **2022**, *314*, 53-80.
22. Tan, H.; Liu, L.; Liang. H.; Total $\{k\}$-domination in special graphs. *Math. Found. Comput.* **2018**, *1*, 255-263.
23. Lee, C.-M. R-total domination on convex bipartite graphs. *J. Comb. Math. Comb. Comput.* **2012**, *81*, 209-224.
24. Lee, C.-M. The complexity of total k-domatic partition and total R-domination on graphs with weak elimination orderings. *Int. J. Comput. Math.: Comput. Syst. Theory*, **2020**, *5* 134-147.

25. Bonomo, F.; Brešar, B.; Grippo, L.N.; Milanič, M.; Safe, M.D. Domination parameters with number 2: interrelations and algorithmic consequences. *Discrete Appl. Math.* **2018**, *235*, 23-50.
26. Brešar, B.; Dorbec, P.; Goddard, W.; Hartnell, B.; Henning, M.A.; Klavžar, S.; Rall, D.F. Vizing's conjecture: A survey and recent results. *J. Graph Theory*, **2012**, *69*, 46–76.
27. Cabrera-Martínez, A.; Conchado Peiró, A. On the $\{2\}$-domination number of graphs. *AIMS Math.* **2022**, *7*, 10731-10743.
28. Cabrera-Martínez, A.; Montejano, L.P.; Rodríguez-Velázquez, J.A. From w-domination in graphs to domination parameters in lexicographic product graphs. *Bull. Malays. Math. Sci. Soc.* **2023**, *46*, 109.
29. Cheng, Y.J.; Fu, H.L.; Liu, C.A. The integer $\{k\}$-domination number of circulant graphs. *Discrete Math. Algorithms Appl.* **2020**, *12*, Article 2050055.
30. Choudhary, K.; Margulies, S.; Hicks, I.V. Integer domination of Cartesian product graphs. *Discrete Math.* **2015**, *338*, 1239-1242.
31. Krop, E.; Davila, R.R. On a Vizing-type Integer Domination Conjecture. *Theory Appl. Graphs*, **2020**, *7*, Article 4.
32. Villamar, I.R.; Cabrera-Martínez, A.; Sánchez, J.L.; Sigarreta, J.M. Relating the total $\{2\}$-domination number with the total domination number of graphs. *Discrete Appl. Math.* **2023**, *333*, 90-95.
33. Zverovich, V. On general frameworks and threshold functions for multiple domination. *Discrete Math.* **2015**, *338*, 2095-2104.
34. Liao, C.; Chang, G.J. Algorthmic aspects of k-tuple domination in graphs. *Taiwan. J. Math.* **2002**, *6*, 415-420.
35. Dobson, M.P.; Leoni, V.; Nasini, G. The multiple domination and limited packing problems in graphs. *Inform. Process. Lett.* **2011**, *111*, 1108-1113.
36. Dobson, M.P.; Leoni, V.; Lopez Pujato, M.I. k-tuple and k-tuple dominations on web graphs. *Mat. Contemp.* **2020**, *48*, 31-41
37. Bellmonte R.; Vatshelle, M. Graph classes with structured neighborhoods and algorithmic applications. *Theor. Comput. Sci.* **2013**, *511*, 54–65.
38. Bui-Xuan, B.; Telle, J.A.; Vatshelle, M. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theor. Comput. Sci.*, **2013**, *511*, 66–76.
39. Barman, S.C.; Mondal, S.; Pal. M. Minimum 2-tuple domianting set of permutation graphs. *J. Appl. Math. Comput.* **2013**, *43*, 133-150.
40. Sinha, A.K.; Rana, A.; Pal, A. The 2-tuple domination problem on trapezoid graphs. *Ann. Pure Appl. Math.* **2014**, *7*, 71-76.
41. Sinha, A.K.; Rana, A.; Pal, A., The 2-tuple domination problem on circular-arc graphs. *J. Math. Inform.* **2017**, *8*, 45-55.
42. Lan, J.K.; Chang, G.J. On the algorithmic complexity of k-tuple total domination. *Discrete Appl. Math.* **2014**, *174*, 81-91.
43. Lee, C.-M. Signed and minus total domination on subclasses of bipartite graphs. *Ars Combinatoria* **2011**, *100*, 129-149.
44. Cormen, T.H.; Leiserson, C.E.; Rivest, R. L.; Stein, C. *Introduction to Algorithms* (4th ed.). MIT Press, 2022.
45. Schrijver, A. *Theory of linear and integer programming*. John Wiley & Sons, 1998.
46. Diestel, R. *Graph Theory* (6th ed.). Springer Berlin, Heidelbergs, 2024.
47. Strang, G. *Introduction to linear algebra*. Wellesley-Cambridge Press, 2022.
48. Rose, D.J. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.* **1970**, *32*, 597-600.
49. Farber, M. Characterizations of strongly chordal graphs. *Discrete Math.* **1983**, *43*, 173-189.
50. Uehara, R. Linear time algorithms on chordal bipartite and strongly chordal graphs. In Automata, Languages and Programming: 29th International Colloquium, ICALP 2002 Málaga, Spain, July 8–13, 2002, pp. 993-1004.
51. Tardella, F. The fundamental theorem of linear programming: extensions and applications. *Optimization* **2011**, *60*, 283-301.
52. Khachiyan, L.G. A polynomial algorithm for linear programming. *Doklady Akademii Nauk SSSR*, **1979**, *244*, 1093-1096.
53. Karp, R.M. Reductibility among combinatorial problems. *Complexity of computer computations*, **1972**, *1*, 85-103.
54. Scheinerman, E.R.; Ullman, D.H. *Fractional Graph Theory: A Rational Approach to the Theory of Graphs.* Dover Publications, INC. Mineola, New York, 2011.

55. Van den Brand, J. A deterministic linear program solver in current matrix multiplication time. In Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2020, 259-278.
56. Cohen, M.B.; Lee, Y.T.; Song, Z. Solving linear programs in the current matrix multiplication time. *J. ACM, 2021* **68**, 1-39.
57. Williams, V.V.; Xu, Y.; Xu, Z.; Zhou, R. New bounds for matrix multiplication: from alpha to omega. In Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Society for Industrial and Applied Mathematics, 2024, pp. 3792-3835.
58. Zhang, F. (Ed.) *The Schur complement and its applications* (Vol. 4). Springer Science & Business Media, 2006.