UNIVERSIDAD DE BUENOS AIRES

Facultad de Ciencias Exactas y Naturales

Departamento de computación

Algoritmos y complejidad para algunos problemas de dominación

Tesis presentada para optar al título de Doctor de la Universidad de Buenos Aires en el área Ciencias de la Computación

Michel Jonathan Mizrahi

Director de Tesis:	Dr. Min Chih Lin
Consejero de Estudios:	Dr. Min Chih Lin
Lugar de trabajo:	Instituto de Cálculo, FCEyN, Universidad de Buenos Aires
Fecha de defensa:	21 de Noviembre, 2014

Buenos Aires, 2014

Algoritmos y complejidad para algunos problemas de dominación

Resumen

Los problemas de dominación forman un área de investigación en crecimiento, debido a la cantidad de aplicaciones que pueden modelar, entre las cuales podemos nombrar redes sociales, sistemas distribuidos, redes biológicas, problemas de localización de instalaciones, etc. En esta tesis estudiamos los siguientes problemas de dominación (i) el conjunto dominante mínimo, (ii) dominación romana, (iii) dominación eficiente por vértices, (iv) dominación eficiente por aristas (también conocida como *matching* inducido dominante), (v) dominación perfecta por vértices (vi) dominación perfecta por aristas, y (vii) subgrafo cordal máximo inducido sin vertices propiamente dominados (también conocido como eliminación de vértices para formar clusters). Para el problema (i) determinamos su complejidad para clases de grafos donde se prohiben subgrafos inducidos con a lo sumo cuatro vértices. Estudiamos los problemas (i) y (ii) para varias subclases de grafos P_5-free, dando algoritmos eficientes, robustos y simples en ambos casos. Algoritmos de complejidad lineal para grafos arco-circulares fueron presentados para los problemas (iv), (v), (vi) usando algoritmos existentes para el problema (iii). Damos tres algoritmos de tiempo exponencial para resolver el problema (iv) en grafos generales. Además, para el problema (iv), presentamos algoritmos de complejidad $O(n)$ restringidos a grafos cordales, dualmente-cordales, bi-convexos, y claw-free. Estudiamos cuatro variantes del problema (vii) y presentamos algoritmos eficientes para todos ellos cuando nos restringimos a grafos de intervalos propios, grafos de intervalos, grafos arco-circulares, grafos de permutación, y grafos trapezoide. Por otro lado, probamos que las cuatro variantes son NP-Dificil para grafos bipartitos. Finalmente, mostramos que dos variantes son NP-Dificil para grafos split, mientras que las otras dos variantes se pueden resolver en tiempo polinomial.

palabras clave: algoritmos, teoría de grafos, conjunto dominante, complejidad computacional

Introducción

Muchos de los problemas interesantes que surgen de aplicaciones en diversas disciplinas son difíciles de computar. La noción de la dificultad de cómputo de un problema fue formalizada por el famoso trabajo [36], donde se funda la teoría de la NP-Completitud.

Dado que la pregunta P=NP sigue abierta, es valioso encontrar maneras de atacar problemas dificiles de computar (también llamados NP-Completos) aunque no sea con algoritmos polinomiales que encuentren la solución óptima.

Una de las maneras de atacar problemas dificiles es con una búsqueda exhaustiva, que consiste en un algoritmo de fuerza bruta que toma decisiones inteligentes de modo de obtener una respuesta lo antes posible. Esto usualmente implicia descartar soluciones candidatas que llevan a resultados no óptimos. Esta técnica da la garantía de encontrar la solución óptima, pero no hay garantías de que el tiempo que necesite el algoritmo sea polinomial. El no tener garantías de tiempo puede ser prohibitivo para determinado tipo de aplicaciones.

Otra manera de atacar problemas es restringir el dominio. Los datos provenientes de aplicaciones que surgen en distintas diciplinas no suelen ser datos arbitrarios, sino que muchas restricciones pueden ser asumidas sobre ellos. Estas restricciones suelen estar reflejadas en propiedades estructurales que los datos deben cumplir. Esto permite restringir el problema a casos donde el cómputo de la solución puede ser llevado a cabo de manera óptima con un algoritmo polinomial. Esta técnica es particularmente usada en el área de teoría de grafos, debido a que los grafos

generales (sin restricciones) permiten modelar estructuras muy generales, y por lo tanto los problemas originales pueden ser representados con grafos que cumplan varias restricciones muy particulares. Esto impulsó el estudio de los grafos restringidos a determinadas estructuras, las cuales fueron formalizadas de muchas maneras, principalmente a través de la representación usando intersección de conjuntos [50]. Esto llevó al estudio de *familias de grafos* definidas por conjuntos, donde cada conjunto representa un vértice de un grafo, y dos vértices son adyacentes si los conjuntos tienen intersección no vacía.

Estas dos técnicas, *búsqueda exhaustiva* y *restricción del dominio* han sido las principales herramientas usadas para atacar los problemas planteados en esta tesis, pero no son las únicas posibles, entre las técnicas alternativas podemos nombrar la *programación lineal entera* [72], *complejidad parametrizada* [46], *algoritmos aproximados* [95], *Monte-Carlo and Las-Vegas methods* [37], y heuristicas como algoritmos genéticos, búsqueda local, entre otras [90].

En esta tesis nos enfocamos en el problema del conjunto dominante de cardinalidad mínima, ampliamente estudiado en teoría de grafos, que consiste en encontrar un conjunto de vértices de mínima cardinalidad tal que cada vértice del grafo pertenezca al conjunto encontrado, o sea adyacentes a algún vértice del mismo.

Durante el segundo capítulo identificamos la complejidad del problema restringiendo el dominio a determinadas familias de grafos. En particular, estudiamos la complejidad para grafos prohibiendo como subgrafo inducido algún grafo de orden menor a cinco.

En el tercer capítulo, hacemos un análisis enfocado en la parte algorítmica del problema, estudiando lo que ocurre para subclases de grafos que tienen prohibido contener a un P_5 como subgrafo inducido. Estudiamos para estas mismas clases de grafos, una variante del problema, denominada *dominación romana*, proveniente de una manera de optimizar la acomodación de legiones del ejército romano, que puede ser fácilmente adaptado a un problema de alocación de recursos. Mostramos

algoritmos extremadamente simples y eficientes para resolver el problema en varias de estas subclases.

En el cuarto capítulo mostramos tres algoritmos exactos para resolver una variante del problema de dominación, donde se pide dominar las aristas del grafo. Los algoritmos usan la técnica de *búsqueda exhaustiva*, pero probamos además que para dos de ellos, dados ciertas restricciones sobre los datos de entrada, el algoritmo se comporta polinomial.

También mostramos como mejorar los algoritmos ya estudiados para resolver este problema en diversas clases de grafos: cordales, dualmente-cordales, biconvexos, y claw-free.

En el quinto capítulo mostramos como resolver dos variantes de dominación ampliamente estudiadas, llamadas *dominación eficiente* y *dominación perfecta* en grafos arco-circulares.

Finalmente en el sexto capítulo planteamos una variante de uno de los problemas más estudiados de clustering, que puede ser descripto a partir de conceptos de dominación de vértices. Damos resultados para este problema cuando se restringen los datos de entrada, en particular, damos algoritmos eficientes para grafos de intervalos-propios, intervalos, arco-circulares, trapezoides. También mostramos la complejidad del problema en grafos split y grafos bipartitos.

Complejidad restringida por subgrafos inducidos prohibidos

En este capítulo estudiamos la complejidad computacional del problema *conjunto dominante de cardinalidad mínima* en grafos caracterizados por subgrafos inducidos prohibidos. Damos una serie de resultados para el problema en grafos definidos para cualquier combinacion de subgrafos inducidos prohibidos con a lo sumo cuatro vértices, resultando en una pruba de NP-Completitud o un algoritmo polinomial para cada caso. Finalmente, extendemos los resultados mostrando que el problema sigue siendo NP-Dificil, incluso para grafos planares, de grado máximo tres, y sin claw, diamantes, K_4 o ciclos de longitud 4,5,7,8,9,19, y 11, todos ellos inducidos.

Comenzamos mostrando algunos resultados conocidos sobre la complejidad del problema, cuando se prohiben algunos subgrafos inducidos de tamaño menor o igual a cuatro [6, 32, 70]. Luego, enumeramos los resultados conocidos, más algunos otros que son fácilmente deducibles, para dar una descripción clara de lo que se conoce hasta el momento, y lo que falta. Mostramos también la información en forma visual, para facilitar la búsqueda de resultados conocidos, y no conocidos. Explicamos cuales resultados deben ser provistos para tener un esquema completo que permita saber, dado cualquier conjunto C de grafos de orden a lo sumo cuatro, cuál es la complejidad del problema en un grafo C-free (es decir, un grafo donde los elementos de C están prohibidos como subgrafos inducidos).

Finalmente, procedemos a hacer las demostraciones necesarias para poder completar el esquema, y extendemos una de esas demostraciones a un resultado más general.

Algoritmos restringidos a grafos P_5-free

El problema de dominación romana mínima es una variante del problema de conjunto dominante mínimo. Ambos problemas son NP-Completos cuando se los restringe a grafos P_5-free (grafos sin P_5 inducidos) [6, 35]. En este capítulo estudiamos ambos problemas para algunas subclases de los grafos P_5-free.

Hemos propuesto algoritmos muy simples para determinar la solución a ambos problemas, de modo que corra en tiempo polinomial si el resultado es una constante. El mismo algoritmo es extendido para resolver el problema eficientemente en cografos y en grafos (P_5,(s,t)-net)-free. Nuestros algoritmos mejoran resultados previos sobre (P_5,bull)-free [70], y (K_p, P_5)-free, con p constante [105]. Si bien ya existían algoritmos eficientes para resolver el problema en cografos [32, 51, 75, 87], todos ellos dependen de estructuras de datos sofisticadas como cotrees, descomposiciones modulares, extensiones homogeneas, etc. o requieren obtener un modelo apropiado del grafo original para luego poder aplicar el algoritmo. Nuestra propuesta es extremadamente simple, y usa los mismos procedimientos que para el resto de los algoritmos.

Dominación eficiente por aristas

En este capítulo desarrollamos algoritmos exactos para la versión por pesos y de conteo del problema de dominación eficiente por aristas, también conocida como Matching Inducido Dominante. Este problema ha sido estudiado de manera extensiva [20, 22, 24, 27, 28, 29, 69, 80, 81]. También desarrollamos algoritmos para el problema restringido a diversas clases de grafos. Este problema está motivado por diversas aplicaciones en las áreas de teoría de códigos, ruteos en redes, y alocación de recursos. Más detalles del problema, asi como sus aplicaciones relacionadas, pueden encontrarse en [61, 78].

Dada una arista $e \in E$, decimos que e se domina a si misma, y a cada arista que comparte un vértice con e. Un subconjunto $E' \subseteq E$ es un matching inducido de G si cada arista de G es dominada por a lo sumo una arista en E'. Un Matching Inducido Dominante de G es un subconjunto de aristas que domina a todas las aristas de G y a la vez, es un Matching Inducido. El problema, en su variante más simple, consiste en identificar si un grafo G contiene algún Matching Inducido Dominante. Este problema es NP-Dificil. Dada una función $E(G) \to \mathcal{R}$, la versión pesada del problema es encontrar el Matching Inducido Domnante tal que la suma de pesos de sus aristas sea mínima. La versión de conteo consiste en contar la cantidad de Matchings Inducidos Dominantes que existen en el grafo. Los algoritmos desarrollados en este capítulo resuelven la versión con pesos y pueden ser adaptados fácilmente para la versión de conteo del problema. El primer algoritmo presentado corre en tiempo lineal, dado un conjunto dominante de cardinalidad constante. El segundo algoritmo

corre en tiempo polinomial, en caso que la cantidad de conjuntos independientes del grafo de entrada sea polinomial. El tercero tiene complejidad temporal $O^*(1.1939^n)$, y complejidad espacial lineal.

Finalmente mostramos como se pueden mejorar algoritmos para resolver este problema grafos cordales, dualmente-cordales, biconvexos y claw-free.

Dominación Perfecta y Eficiente

En el capítulo anterior mostramos algoritmos para el problema de dominación eficiente por aristas (también conocido como Matching Inducido Dominante). Este problema proviene de un problema más general, que son la dominación eficiente y perfecta. Dado un grafo $G = (V, E)$, un conjunto dominante perfecto es un subconjunto de vértices $V' \subseteq V(G)$ tal que cada vértice $v \in V(G) \setminus V'$ es dominado por exactamente un vértice $v' \in V'$. Un conjunto dominante eficiente es un conjunto dominante perfecto V', tal que V' es un conjunto independiente. Todo grafo G contiene un conjunto dominante perfecto, por ejemplo, $V(G)$ es un conjunto dominante perfecto para cualquier grafo G. Pero no todo grafo contiene un conjunto dominante eficiente. Los problemas consisten en minimizar la cantidad de vértices de los conjuntos dominantes (perfecto y/o eficientes). Estos problemas, incluso restringidos a familias de grafos, son NP-Dificil. La versión pesada consiste en un grafo donde cada vértice v tiene un peso, y se quiere minimizar la suma de pesos de los vértices.

Todo lo dicho anteriormente puede ser aplicado en aristas, en lugar de vértices. En el caso de dominación eficiente por aristas, el problema es equivalente al Matching Inducido Dominante.

En este capítulo, mostramos algoritmos para los cuatro problemas, cuando se restringe a grafos arco-circulares.

Eliminación de vértices para formar clusters

Muchos problemas de clustering consisten en identificar objetos similares en categorías, y excluir los objetos que no pertenecen a ningúna de ellas. Esos elementos pueden provenir de errores de medición, o pueden ser simplemente elementos fuera de la norma. El objetivo es erradicar esos elementos de modo que el resultado sean grupos de elementos bien categorizados. Muchas aplicaciones surgen de obtener clusters a partir de estudiar la relación entre cada par de objetos [4, 5, 45].

En este capítulo estudiamos el problema de eliminación de vertices para formar clusters. El problema consiste en obtener la mínima cantidad de vértices que deben ser eliminados de modo que el grafo resultante sea una unión disjunta de cliques, también conocido como grafo *cluster*. Presentamos algoritmos polinomiales para el problema en sus cuatro variantes: con o sin pesos, y con o sin número fijo de cliques. En el caso de la versión con pesos, el problema consiste ahora en un grafo con pesos en los vértices, y se quiere minimizar la suma de los pesos de los vértices eliminados, en lugar de su cardinalidad. En el caso del número fijo de cliques, el problema contiene un parámetro adicional que restringe a la cantidad exactas de cliques disjuntas que debe tener el grafo *cluster* resultante. Es importante destacar que el caso pesado es más general que la variante sin pesos (pués es equivalente a resolver el problema utilizando peso 1 en cada vértice). Del mismo modo, la variante que establece una cantidad exacta de cliques (representado por un parámetro que denotamos D) para el grafo

resultante, es más dificil que la versión que no esta restringida por este parámetro, debido a que resolver el problema con todos los parámetros posibles desde $D = 1$ hasta $D = n$ sirve para encontrar la solución de la versión sin parámetros.

Estudios anteriores analizaron este problema en términos de algoritmos parametrizados [13, 17].

Mostramos algoritmos de complejidad temporal y espacial polinomial para las cuatro variantes del problema en diversas clases de grafos, entre ellas, intervalos-propios, intervalos, arco-circulares, permutación y trapezoides.

Luego, mostramos que la variante sin cliques fijas para grafos split se puede resolver en tiempo polinomial, mientras que al agregar el parámetro D, el problema se vuelve NP-completo.

Finalmente damos una demostración de que el problema en su variante más sencilla es NP-Completo para grafos bipartitos, en consecuencia, el problema es dificil para todas sus variantes.

Conclusiones

En esta tesis presentamos nuevos resultados en diversos problemas de dominación. Además, pusimos énfasis en hacer muchos de los algoritmos descriptos útiles para fines prácticos, es decir, simplificando los algoritmos y las estructuras de datos usadas, manteniendo la misma complejidad temporal y espacial.

Al comienzo de la tesis, mostramos algunos resultados teóricos interesantes, dando un resultado de complejidad para el problema general en diversas familias de grafos muy estudiadas.

Luego mostramos una variante, llamada dominación romana, y vimos como algunos algoritmos sencillos pueden servir para resolver el problema de dominación, y de dominación romana para subclases de grafos P_5-free. Una gran ventaja en este caso es la simpleza de los algoritmos, que luego son usados para dar un nuevo algoritmo para cografos, que no mejora la performance de los anteriores, pero simplifica mucho su implementación y demostración.

En el siguiente capítulo, mostramos diferentes maneras de atacar el problema de dominación eficiente por aristas. Mostramos algoritmos exponenciales no triviales que sirven para resolver el problema en grafos generales, y demostramos que bajo ciertas circunstancias, los algoritmos encuentran el resultado en tiempo polinomial.

Adicionalmente, estudiamos los problemas de dominación perfecta y eficiente por vértices y aristas, problemas que forman casos más generales que lo hecho en el capítulo anterior. En este sentido, damos los resultados para estos cuatro problemas en grafos arco-circulares.

El problema de Cluster-Vertex-Deletion es estudiado en términos de restricciones en el dominio de entrada, esto es, restringido a determinadas familias de grafos. Mostramos resultados interesantes para el problema en varias de las familias de grafos más estudiadas, y presentamos algoritmos de complejidad óptima en algunos casos, así como resultados de su complejidad en los casos que encontramos el problema es NP-Completo.

Algorithms and complexity for some domination problems

Abstract

Domination is a growing research area in graph theory, with a vast number of applications across different disciplines, which include social networks, distributed computing, biological networks, facility location problems, etc. In this thesis we studied the following domination problems (i) minimum dominating set problem (ii) Roman domination (iii) efficient vertex domination (iv) efficient edge domination (also known as dominating induced matching or DIM) (v) perfect vertex domination (vi) perfect edge domination (vii) maximum induced chordal subgraph without properly dominated vertices (also known as cluster vertex deletion). For problem (i) we determined its complexity for graph classes defined by forbidding as induced subgraphs all graphs with at most four vertices. We studied problems (i) and (ii) for some subclasses of P_5-free graphs, giving efficient, robust and simple algorithms for both of them. Linear time algorithms restricted to circular-arc graphs were presented for problems (iv), (v), (vi) using existent linear algorithms from problem (iii). We described three exact exponential time algorithms solving problem (iv) for general graphs. Also, for problem (iv), $O(n)$ time algorithms were given restricted to chordal, dually-chordal, bi-convex and claw-free graphs. We studied four variants of problem (vii) and presented efficient algorithms for all variants whenever the graphs were proper-interval graphs, interval graphs, circular-arc graphs, permutation graphs and trapezoid graphs. On the other hand we proved that the four variants are NP-Hard for bipartite graphs. Finally we showed that two of the variants are NP-Hard for split graphs while the other two variants are polynomially solvable.

keywords: algorithms, graph theory, dominating set, computational complexity

Agradecimientos

Primero y principal, quiero agradecer a mi director, Oscar (Min Chih Lin), ya que sin él nada de esto hubiese sido posible. Me siento extremadamente afortunado de haberlo tenido como director, no solo aprendí como alumno, sino también como persona. Oscar me abrió las puertas desde el primer día, desde el primer email comentándole sobre mi interés en investigación, cuando aún no tenía definido si iba a hacer un doctorado con él. Con los años confirmé que es excelente como persona, y como director. Me supo guiar durante todo momento, y me dió la libertad de explorar, buscar e investigar sobre temas y líneas de investigación que yo creía interesantes. Me puso una prioridad que nunca hubiese esperado, gracias a lo cual esta tesis prosperó, y además hizo lo imposible para asegurarse que pueda viajar y presentar los resultados en congresos internacionales, lo cual fue esencial en mi formación. Gracias!

A Jayme Szwarcfiter, con quien trabajé durante gran parte de mi tesis. Agradezco el conocimiento que me supo transmitir, y la humildad con la cual lo hizo. La estadía en Brasil para trabajar junto a él fue más que gratificante, no sólo aprendí mucho trabajando frente a una persona de semejante trayectoria académica, sino que también aprendí lo que es ser una gran persona en todos los aspectos. Gracias por recibirme en tu casa, y gracias por todas las enseñanzas en todos los ámbitos.

A mi familia, Marvin, Pa y Ma, les agradezco eternamente por su amor incondicional.

A Dan, porque sin él nunca hubiese conocido esta facultad. A Mich, por ser de esos valiosos amigos que sabes que siempre está cuando lo necesitas. Al grupete

de siempre, Maru, Pablox, Fer, Luigi, Nahuel, Ale, y Vicky con quien compartí una inumerable cantidad de salidas, anécdotas y vivencias, que hicieron este camino mucho más divertido. A mis compañeros de oficina porque disfruto hablar y actualizarme de todo, las pocas veces que voy a la oficina. A Saveli, porque es de las pocas personas con las que puedo me quedar hablando de política o filosofía, hasta las 4 am.

A todos los que fueron compañeros, ayudantes, estudiantes, docentes, o amigos porque ayudaron a mi formación en todos los aspectos de mi vida.

Por último, y no menos importante, quiero agradecer al CONICET por la beca que me permitió realizar esta tesis. Al Instituto de Cálculo por brindarme un lugar ameno donde trabajar, al Departamento de Computación (FCEyN) de la Universidad de Buenos Aires, porque es a quien debo mi formación como estudiante, docente e investigador.

I want to add special thanks to the jury members for their thorough review, which helped to improving the quality of this thesis.

Contents

Chapter 1

Introduction

Many of the interesting problems that arise from applications in different disciplines have been proved to be hard to compute. The hardness of the problems were formalized by the seminal paper [36], which laid the foundations for the theory of NP-Completeness. It became clear that the hard questions that many computer scientists were trying to solve belong to the NP-Complete class, and pose the question of P=NP. Actually no one knows if the hardest problems from the NP class can be computed in polynomial time on the input size, but many people assumes it is highly improbable to solve these problems in an optimal way within a reasonable time (i.e. polynomial time). Moreover, even if this can be accomplished in the future, no one is expecting a practical polynomial time universal algorithm that behaves well for problems from the whole NP class. Therefore it is valuable to be able to solve many of those problems now with the current tools and knowledge that is available. The fact that it is not currently possible to solve them in a reasonable time lead to different approaches in order to give solutions that may serve for practical purposes. One of these approaches is the *intelligent exhaustive search*, which consists on a brute force search algorithm that makes some smart decisions in order to obtain an answer as soon as possible. This usually involves discarding candidate solutions that lead to non optimal results, which ultimately saves computing time. This technique is usually

referred as *Backtracking* or *Branch & Reduce*. It guarantees that the optimal solution will be found, which makes it a useful approach for certain applications. The main drawback of an exhaustive search approach is that it gives no guarantee that the time it needs to find the optimal solution is bounded by a polynomial function in the size of the input, that is, the worst case of the algorithm could take an exponential number of steps (in size of the input), which may be an unreasonable amount of time. This is not acceptable for many problems where certain time guarantees are needed.

Another common approach is to restrict the problem domain. It turns out that many times the input is not arbitrary data, and a lot of assumptions over the input instances can be made according to the field where the problem came from. These restrictions come often in the form of structural properties that each input instance must satisfy. The structural properties restrictions have been widely studied, particularly in graph theory. It is known that graph theory is useful to model problems, to the point that it is even hard to come up with problems that cannot be modeled with graph theory. The fact that graph theory is so powerful is also a weakness per se, since solving problems on graphs means solving problems for so many different models that it may be very hard to deal with all of them. Most of the times, the problem we want to solve may be able to be modeled with certain restricted graphs. This was the case for many interesting problems, which lead to the interesting field of studying graph problems restricted to certain graph structures. These structures became formalized in many ways, one of the most commonly used being the representation of graphs using set intersections [50] which lead to *graph families* defined by the possible sets to be used (also known as *graph classes*). The number of graph families have grown exponentially over the last decades [21] with hundreds of graph classes being studied nowadays. Many of these classes arise from practical interests, and many others from theoretical interests. Among graph classes with practical interests we can name the interval graphs [21, 74], circular-arc graphs, [21, 58], permutation graphs [3, 21], split graphs [21, 84], bipartite graphs [1, 21], all of which have been studied in this thesis.

These approaches were mainly used at this thesis, but they are not the only ones that exist. The following are widely used, but this thesis makes no use of them.

Techniques such as *branch and bound* and *integer programming* [72] rely on the same principles of *intelligent exhaustive search*, by making smart decisions in order to minimize the time needed to find the solution. They make a heavy use of linear algebra language in order to pose the problems and the solutions.

The related parameterized complexity field has grow interest recently, where the algorithm may behave within exponential time on some parameters of the instance instead of the total size of the input. The complexity of the problem can be measured as a function of those parameters. A foundational book on the topic is [46].

Alternative approaches include the probabilistic algorithms such as Monte-Carlo method whose running time is deterministic but whose output may be incorrect with a certain probability [37], and Las-Vegas [37] which is a randomized algorithm whose running time is not deterministic, this is, it may take very long time with a certain probability, but the answer is deterministic and always correct. One of the most popular approaches is that of approximation algorithms [95]. The method sacrifices the optimality of the solution in order to obtain a polynomial time algorithm, nonetheless the solution is within a certain threshold from the optimal solution, thus the solution is no worse than some guarantee that the method gives.

Note that previous approaches give some guarantee of time complexity or solution correctness, but there is a basic trade-off that cannot be surpassed unless P = NP. The alternative approach is the use of heuristic methods such as local search, simulated annealing, genetic algorithms, etc. [90] which give no guarantee of runtime or correctness of the solution. It turns out that many times the problems from real life are so messy and hard to deal with, that to ensure guarantees of time or correctness there is a need of time and resources which are not usually available, or even worst, no one could give any guarantees for those problems yet. Hence the most practical solution for these cases is to use heuristics which may give very good results, in short time and may be easier to implement.

1.1 Background

The graph theory discipline started several centuries ago, many referring to Leonard Euler being one of the founders by studying the problem of the seven bridges of Köningsberg [7], being this the first paper that effectively uses graph-theoretic ideas. A graph is a mathematical structure that serves as a representation of relations among different objects. The usual convention is to say the objects are vertices (represented by points) and the relations are edges (represented by lines that joins those points). The graphs are useful as a tool for modeling problems that come from very different sources, allowing to use the same theoretical properties to solve problems that may seem totally unrelated. Ultimately many problems can be described by a set of relations among different elements or processes, as may be a social network described by people and their friendship or biological data where the elements may be different DNA data and the relations are among similar elements (same specie data). There is no restriction on the objects and their relationship, this is, a graph can be any set of objects and any two objects may be related or not, even an infinite number of objects. It turns out problems from real life usually do not contain infinite elements, and moreover, usually can be modeled with graphs which satisfy several properties that impose a lot of restrictions to the represented graphs. This leads to the study of graphs defined by set intersection [50], which serve to restrict the number of models that can be represented, as we already mentioned before. The advantage of having graphs that are more restricted and can represent only a restricted subset of models is that it is easier to work on solutions for these restricted representations. An overview of the intersection graph theory can be found in [21].

Many of the problems that are posed in terms of graph theory seem to appear repeatedly, such is the case of the minimum dominating set problem. The *minimum dominating set* problem along with some variations emerges from numerous problems on diverse areas. A good source with several applications of domination problems can be found in [89]. The minimum dominating set problem, along with many of the

interesting variations of this problem, was proved to be NP-Complete [57]. It was also shown that it is polynomial time approximable with factor $1 + \lg |V|$ using it as a special case of the minimum set cover [65], however, it is not approximable within $(1 - \epsilon) \ln |V|$ for any $\epsilon > 0$ unless $NP \subseteq DTIME$ [52]. The problem can be solved by a polynomial-time algorithm when the input domain is restricted, for instance, for graph families such as interval graphs [30], permutation graphs [32], trees, etc. These facts lead to an interesting field of study which consists on identifying the domains where the instances can be solved in polynomial-time. The restrictions on the input are usually studied by restricting the study to certain graph families. In the next section, we give the formal definitions of the problems studied on this thesis, along with the corresponding references that point the motivations and past work done on them.

1.2 Notations and Definitions

A graph is an ordered pair $G = (V, E)$ comprising a set V of vertices or nodes together with a set E of edges or lines, which are 2-element subsets of V. An edge gives the relation between two vertices, which is represented by the pair of vertices. In the case of directed edges, an edge is represented by an ordered pair and we say the graph is *directed*, otherwise it is represented by an unordered pair and we say the graph is *undirected*. Unless stated otherwise, along this thesis we consider undirected graphs G, denoting by $V(G)$ and $E(G)$, respectively, the sets of vertices and edges of G, $n = |V(G)|$ and $m = |E(G)|$. For $v \in V(G)$, $N(v)$ represents the set of neighbors of $v \in V(G)$, while $N[v] = N(v) \cup \{v\}$. For $S \subseteq V(G)$, $N(S) = \cup_{v \in S} N(v) \setminus S$, and $N[S] = \cup_{v \in S} N(v) \cup S$. We say a vertex $v \in V(G)$ such that $N[v] = V(G)$ is *universal*, and a *matching* is a set of pairwise non-adjacent edges.

A *path* is a sequence of consecutive edges in a graph. Unless is stated otherwise, we assume a path is simple, meaning that no vertices are repeated. The *length* of a path is the number of edges it contains. The *distance* between two vertices u and v

in a graph G, is the minimum length among all paths from u to v. A *cycle* is a *path* where the first and the last vertex are the same. A *chord* of a path or a cycle is an edge that joins two non-consecutive vertices of the path or cycle. As usual, C_n and P_n denote the chordless cycle and the chordless path on n vertices.

A *dominating set* for a graph $G = (V, E)$ is a subset $D \subseteq V$ such that every vertex not in D is adjacent to **at least** one vertex of D. The domination number $\gamma(G)$ is the number of vertices in a smallest dominating set for G. The dominating set problem concerns testing whether $\gamma(G) \leq K$ for a given graph G and input K. The first papers about this problem can be dated from 1950, as noted by [63]. The problem is NP-Complete [57].

We say an *induced subgraph* is a subset of the vertices of a graph G together with any edges whose endpoints are both in this subset. Given a graph $G = (V, E)$ and a subset of the vertices $V' \subseteq V$, we denote with $G[V']$ the subgraph induced by the subset V'.

An induced subgraph H of G is called *dominating* if $V(H)$ is a dominating set of G. We say a vertex of degree one is a *pendant* vertex. A graph is said to be *connected* if there is a path joining u and v, for any two vertices u and v, and a *connected component* is a maximal connected subgraph.

A *complete set* of a graph is a set of pairwise adjacent vertices, while a *clique* is a maximal *complete set*. The complete graph of order n is denoted by K_n. A graph that can be partitioned into two set of vertices V_1, V_2 such that each vertex of the V_1 is connected to each vertex of V_2, and no adjacencies exists between vertices in V_1 and V_2, is denoted as $K_{n,m}$, where $|V_1| = n$ and $|V_2| = m$. The graph K_4 minus one edge is called *diamond*, the graph $K_{1,3}$ is called *claw*. The graph that consists of a K_3 plus one vertex connected to exactly one vertex from the K_3 is called *paw*. Whenever the prefix *co* is added, it means the complement of the graph.

Subdivision of an edge is the operation of creation of a new vertex on the edge. When a polynomial-time algorithm has been shown for a problem, we say that the

problem is in P. Whenever the problem is in the complexity class NP-Complete we say that the problem is NPC.

The following formal definition of intersection graphs from [83] is useful to understand several graph classes used along this thesis. Let $\mathcal{F} = \{S_1, \ldots, S_n\}$ be any family of sets. The *intersection graph* of $\mathcal{F}$, denoted $\Omega(\mathcal{F})$, is the graph having $\mathcal{F}$ as vertex set with set S_i adjacent to S_j if and only if $i \neq j$ and $S_i \cap S_j \neq \emptyset$. A graph is an *intersection graph* if there exists a family $\mathcal{F}$ such that $G \cong \Omega(\mathcal{F})$ where we typically display this isomorphism by writing $V(G) = \{v_1, \ldots, v_n\}$ with each v_i corresponding to S_i, thus $v_i v_j \in E(G)$ if and only if $S_i \cap S_j \neq \emptyset$. When $G \cong \Omega(\mathcal{F})$, $\mathcal{F}$ is then called a *set representation* of G.

We define the operation $L : g \to g$ that takes a graph and returns a graph. Given a graph G, we say $L(G)$ is the line graph of G. Given a graph G, the result of $L(G)$ is the intersection graph of the edges of G.

We define the *square* operation over a graph. The square graph G^2 of a graph G is the graph G with the added edges between any two vertices whose distance in G is two.

An *independent set* or *stable set* for a graph $G = (V, E)$ is a set $I \subseteq V$ such that for every two vertices in I, there is no edge connecting the two. Equivalently, each edge in the graph has at most one endpoint in I. The size of an independent set is the number of vertices it contains. A maximum independent set is an independent set of largest possible size for a given graph G. Determining the size of an independent set of maximum cardinality is NPC [57].

The weighted version of both problems deals with a weighted graph, that is, a graph $G = (V, E)$ and a function $w : V \to \mathcal{R}^+$, where each vertex is assigned a weight. The case where the weights of all vertices are exactly 1 is exactly the same as the unweighted version.

A *perfect dominating set* for a graph $G = (V, E)$ is a subset $D \subseteq V$ such that every vertex not in D is adjacent to `exactly` one vertex of D.

An *efficient dominating set* is a *perfect dominating set* D such that D is also an independent set.

Given an edge $e \in E$, say that e *dominates* itself and every edge sharing a vertex with e. A set of edges $E' \subseteq E$ from the graph $G = (V, E)$ is *dominating* if E' dominates every edge from E. Subset $E' \subseteq E$ is an *induced matching* of G if each edge of G is dominated by at most one edge in E'. A *dominating induced matching (DIM)* of G is a subset of edges which is both dominating and an induced matching. The problems of finding a subset of edges $E' \subseteq E$ from a graph $G = (V, E)$ such that E' is an *efficient edge domination* or a *dominating induced matching* are equivalent. Both problems require to identify if such a subset of edges satisfying those properties exists and have been studied using both names.

We say an algorithm is *robust* if its output is correct even if the input does not belongs to the restricted domain. Thus, whenever this is the case, the algorithm can identify that the input is invalid and report it.

The *arboricity* of an undirected graph is the minimum number of forests into which its edges can be partitioned. Equivalently it is the minimum number of spanning forests needed to cover all the edges of the graph.

Graph classes

Along the thesis we work on several graph classes. We provide concise definitions here:

- *AT-free*: Three vertices of a graph form an asteroidal triple if every two of them are connected by a path avoiding the neighborhood of the third. A graph is AT-free if it does not contain any asteroidal triple.
- *Interval*: [100] It is the intersection graph of a family of intervals on the real line.
- *Proper Interval*: An interval graph in which no interval properly contains another.

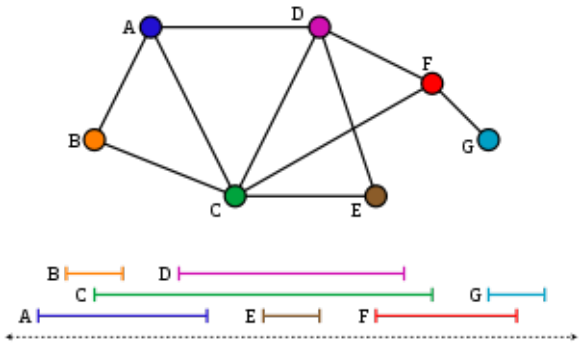

Figure 1.1: Seven intervals on the real line and the corresponding seven-vertex interval graph.

- *Bipartite*: Its vertex set can be partitioned into two independent sets.

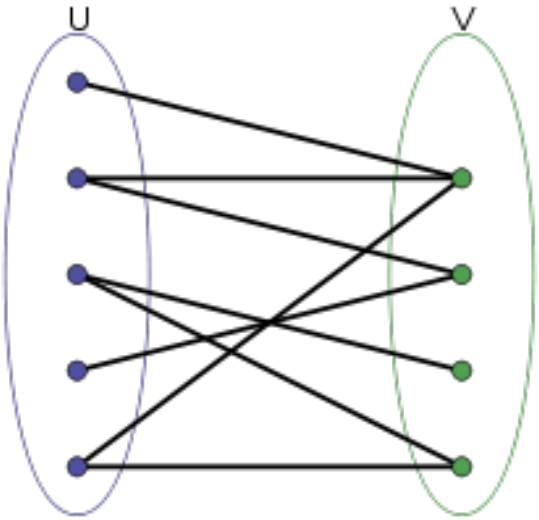

Figure 1.2: Example of a bipartite graph without cycles. [98]

- *Circular-Arc*: It has an intersection model consisting of arcs of a circle.
- *Chordal*: Every cycle of length at least four has a chord.
- *Chordal-Bipartite*: A bipartite graph such that each cycle of length at least six has a chord.
- *Clique-Graph*: The clique graph of a given graph G, denoted with $K(G)$ is the graph intersection of the family of cliques of G.

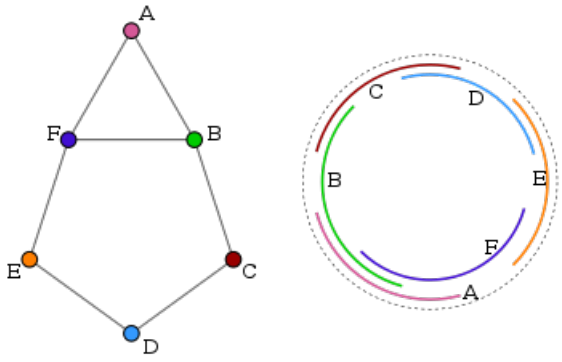

Figure 1.3: A circular-arc graph (left) and a corresponding arc model (right). [99]

- *Cograph*: A graph that does not contain a P_4 as an induced subgraph.
- *Permutation*: It is an intersection graph of a family of straight lines (one per vertex) between two parallels. Equivalently, is a graph whose vertices represent elements of a permutation, and whose edges represent pairs of elements that are reversed by the permutation.

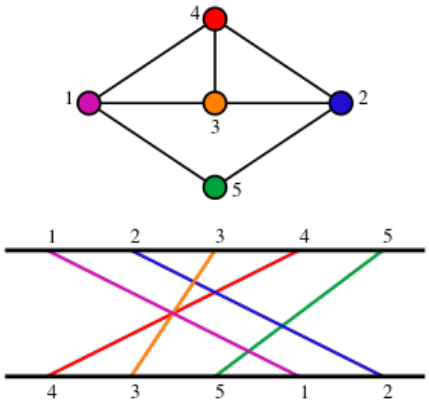

Figure 1.4: The permutation (4,3,5,1,2) and the corresponding permutation graph. [101]

- *Planar*: It can be embedded in the plane (drawn with points for vertices and curves for edges) without crossing edges.
- *Split*: Its vertex set can be partitioned into an *independent set* and a *complete set*.

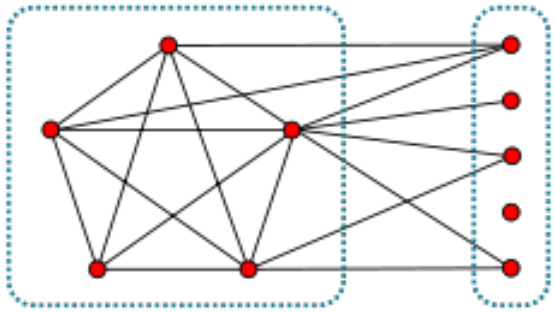

Figure 1.5: A split graph, partitioned into a clique and an independent set. [102]

- *Line*: The line graph $L(G)$ of a graph G is the result of applying the L operation, as mentioned above.
- *Square*: The square graph G^2 of a graph G is the result of applying the ^{2} operation, as mentioned above.

1.3 Overview

Many of the graph families can be defined by forbidden induced subgraphs. Moreover, it has been shown that a few restrictions of these kind makes the behaviour of many NP-Hard problems change completely. For instance, almost all classical problems that are NP-Hard for general graphs can be solved in linear time for cographs.

During the second chapter of this thesis, we study the problem of identifying in which complexity class the domination problem falls when restricted to an $\mathcal{F}$-free family, where $\mathcal{F}$ is the family of graphs to be restricted. We have to restrict ourselves to an affordable size for the family $\mathcal{F}$. The results given allow to identify the class complexity in which the dominating set problem falls, for whichever family

$\mathcal{F}$ conformed by graphs with at most four vertices. Some results were already known but did not allow to establish the complete scheme of which graph restrictions make it possible to establish if the problem falls in either NP-Complete or Polynomial-Time class. Moreover, the results are clearly presented in order to allow a reader an immediate answer (i.e. it does not consists of a huge table for any possible graph family). Given the class complexity is important to know how to approach the problem, but even it may be in P, it does not mean it is easy to deal with.

During the third chapter, we make an analysis focused on the algorithmic side of the problem. It has been shown that the dominating-set problem restricted P_5-free graphs is NP-Hard, while the problem can be solved in linear time for P_4-free graphs, hence it seemed interesting to study the problem for subclasses of P_5-free graphs. There is a barrier between P_4-free and P_5-free graphs that makes the problem behavior much harder. Recall that the motivation for the study of the dominating-set problem arise from the many applications that can be solved using it. Moreover, many of these applications may have some variants, which can be posed as a variant of the dominating-set problem. We show one particular application with a natural variant that originates from it, both of them studied on the third chapter.

There are several variants of the dominating set problem which seem to be easier, such as the efficient edge dominating set (also known as dominating induced matching). We also refer to this problem as DIM. This problem caught more attention and was widely studied [20, 22, 24, 27, 28, 29, 61, 69, 80, 81]. Applications from this problem to coding theory, network routing and resource allocation can be found in [61, 78]. Even this is an NP-Hard problem, it can be solved in linear time, even for P_7-free graphs [24].

Given that this problem seemed to be easier, it was a good candidate to try *Branch & Reduce* technique. The fourth chapter shows three different exact algorithms for the DIM problem (weighted version and counting version of the problem). Of those three algorithms, two of them have polynomial running time whenever certain properties are satisfied (contains a dominating set of fixed size, or contains a polynomial number

of maximal independent sets). The third one has not an identified property for which it behaves as a polynomial-time algorithm but the theoretically upper-bound for the running time is better than the two others. In this chapter we also approach the problem by restrictions on the domain (i.e. for certain graph families). We give bounds for chordal, dually-chordal and biconvex graphs, which leads to improvement of the current algorithms from $O(n+m)$ to $O(n)$. In addition, an improvement from an $O(n^2)$ time algorithm for the unweighted DIM problem on claw-free graphs was improved to an $O(n)$ time algorithm that also solves the weighted-version.

Given two vertices u, v, if $N[u] \subset N[v]$, we say u is *properly dominated* by v, since any dominating set containing u can be replaced by one that uses v instead of u and the cardinality will be less or equal. A chordal graph without properly dominated vertices is a disjoint union of cliques (also known as *cluster graph*). Given a graph $G = (V, E)$, the minimum number of vertices that should be removed in order to obtain a cluster graph $G' = (V', E')$, where $G' = G[V']$ is known as the Cluster-Vertex-Deletion, an NP-Hard problem. It is closely related to the widely studied problem Cluster-Editing (also known as Correlation-Clustering). It turns out that many problems relay on clustering information, therefore, the importance of studying and analyzing clustering algorithms. The cluster vertex deletion problem has not been largely studied, but seems to be a natural way to deal with noise in information. In the fifth chapter we show several algorithms for the problem when restricted to several graph classes such as: Proper-Interval, Interval, Circular-Arc, Permutation and Trapezoid . We also show hardness of the problem for Split graphs, and Bipartite graphs.

Finally, the last chapter shows the summary of the thesis, where all the results are listed.

Chapter 2

Complexity restricted by forbidden induced subgraphs

In this chapter, we study the computational complexity of the minimum dominating set problem on graphs restricted by forbidden induced subgraphs. We give a series of dichotomy results for the problem on graphs defined by any combination of forbidden induced subgraphs with at most four vertices, resulting in either an NP-Hardness proof or a polynomial time algorithm. Finally we extend the results by showing that dominating set problem remains NP-hard even when the graph has maximum degree three, is planar and with no induced claw, induced diamond, induced K_4 nor induced cycle of length 4,5,7,8,9,10 and 11.

2.1 Preliminaries and Previous Results

We present some previous results of the complexity for the minimum dominating set for $\mathcal{F}$-free graphs, where $\mathcal{F}$ is a family of graphs of order at most four. For an unary family $\mathcal{F} = \{F\}$, usually we denote F-Free instead of $\mathcal{F}$-free. Recall that the dominating set problem can be solved independently for each connected component,

hence we assume graphs are connected whenever is convenient. The set of graphs of order three are: P_3, K_3, $3K_1$ and co-P_3.

Lemma 2.1. *[32] The domination problem restricted to permutation graphs is in P.*

Since the class of P_4-free graphs is a subclass of the class of permutation graphs, we obtain:

Corollary 2.1. *The domination problem restricted to the class of P_4-free graphs is in P.*

Lemma 2.2. *[6] The domination problem restricted to bipartite graphs is in NPC.*

Since the class of K_3-free graphs is a superclass of the class of bipartite graphs, we obtain:

Corollary 2.2. *The domination problem restricted to the class of K_3-free graphs is in NPC.*

Lemma 2.3. *[70] The domination problem restricted to AT-free graphs is in P.*

Since the classes of co-paw-free graphs and $3K_1$-free graphs are subclasses of the class of AT-free graphs, we obtain:

Corollary 2.3. *The domination problem restricted to the co-paw-free graphs and $3K_1$-free graphs are in P.*

Lemma 2.4. *[6] The domination problem restricted to split graphs is in NPC.*

Since the class of $(2K_2, C_4)$-free graphs is a superclass of the class of split graphs, we obtain:

Corollary 2.4. *The domination problem restricted to the class of $(2K_2, C_4)$-free graphs is in NPC.*

Lemma 2.5. *[105] The domination problem restricted to (K_p, P_5)-free graphs for fixed p is in P.*

Corollary 2.5. *The domination problem restricted to $(2K_2, K_4)$-free graphs is in P.*

Lemma 2.6. *[15] The clique-width of (claw,co-claw)-free graphs and (claw,paw)-free graphs is bounded.*

Lemma 2.7. *[39] The clique-width of ($K_2 \cup$ claw, K_3)-free graphs is bounded.*

Lemma 2.8. *[38] The domination problem is in P for graph classes with bounded clique-width.*

Corollary 2.6. *The domination problem restricted to (claw,co-claw)-free graphs or (claw,paw)-free or ($K_2 \cup$ claw, K_3)-free graphs is in P.*

We add some remarks for easy results for which we could not find any references:

Remark 2.1. *Note that (claw,K_3)-free $\subseteq$ ($K_2 \cup$ claw, K_3)-free graphs. Hence the domination problem on (claw,K_3)-free graph class is in P.*

Remark 2.2. *The minimum dominating set problem restricted to $4K_1$-free graphs is in P.*

Proof. Given a $4K_1$-free graph G, any maximal independent set I of G has at most 3 vertices. It's well-known that every maximal independent set is a dominating set, then there is at least one dominating set of size at most 3. Hence, the minimum dominating set for G can be solved by checking for each possible subset of at most three vertices if it is a dominating set. This can be done in $O(n^3)$. □

Remark 2.3. *The minimum dominating set problem restricted to co-diamond-free graphs is in P.*

Proof. Given a co-diamond-free graph G, if G has no edge then $\gamma(G) = n$. Otherwise, let $e = v_1v_2$ be an arbitrary edge. If $\{v_1, v_2\}$ is not a dominating set then exists a vertex w such that $\{v_1, v_2\} \cap N(w) = \emptyset$. Clearly, $\{v_1, v_2, w\}$ is a dominating set because if exists some vertex $x \in V(G)$ such that is not adjacent to any of them, then $\{v_1, v_2, w, x\}$ induces a co-diamond which is a contradiction. □

We enumerate existent results for the complexity of the problem restricted to $\mathcal{F}$-free graphs where $\mathcal{F}$ is a family of graphs with 3 or 4 vertices:

Forbidden Induced Subgraphs	Complexity	References
P_3	P	[32]
$3K_1$	P	[70]
co-P_3	P	[70]
P_4	P	[32]
$4K_1$	P	Remark 2.1
co-diamond	P	Remark 2.2
paw,claw	P	[15, 38]
claw,co-claw	P	[15, 38]
co-paw	P	[70]
claw, K_3	P	[38, 39]
K_3	NPC	[104]
claw	NPC	[59]
paw	NPC	[86]
diamond	NPC	[86]
co-claw	NPC	[86]
$2K_2, C_4$	NPC	[6]

We represent certain information contained in above table as a meta-graph where each vertex corresponds to an F-free graph (a vertex gets rectangle shape if the domination problem restricted to its corresponding class is in P; otherwise, it gets circular shape which means the domination problem restricted to its corresponding class is in NPC) and each edge vw corresponds to an intersection of NPC-classes corresponding to v and w such that the intersection is in P. In next section, we will complete the meta-graph with new edges. This allows to determine the complexity

of domination problem restricted to any $\mathcal{F}$-free graphs where $\mathcal{F}$ is a family of graphs with at most 4 vertices.

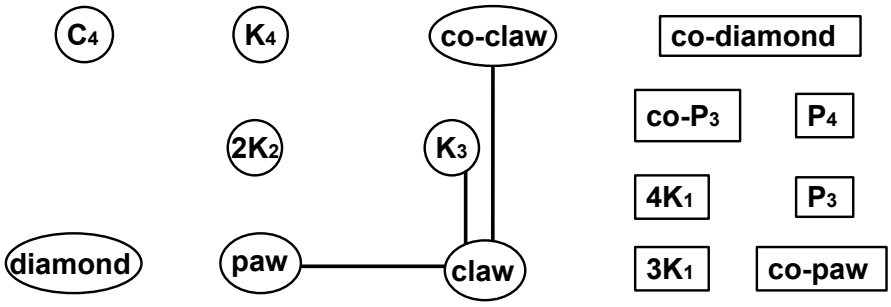

Figure 2.1: Scheme of graph classes complexity for the dominating set problem. Square-shape means P and Circle-shape NPC

2.2 Completing hierarchy

We complete the study of the domination problem by restricting induced subgraphs of order at most four, in the sense that the complexity of the domination problem can be determined for any $\mathcal{F}$-free graph where $\mathcal{F}$ is a family of graphs with at most 4 vertices.

In order to complete Figure 1, we need to prove the complexity of the domination problem for the intersection of classes in which it belongs to NPC. Note that any connected graph which is K_3-free and has at least four vertices is also (K_4,paw,diamond)-free graph. We know that the problem is in NPC for K_3-free graphs, thus it is NPC for (K_4,paw,diamond)-free graphs, and the vertices corresponding to K_4-free, paw-free and diamond-free graphs form an independent set.

Theorem 2.1. *The domination problem restricted to* ($2K_2$,*claw)-free graphs class is in* P.

Proof. Let $G = (V, E)$ be a connected ($2K_2$,claw)-free graph. Since the domination problem for P_4-free graphs is in P, then we consider the case where G has an induced $P_4 = \{p_1, p_2, p_3, p_4\}$ (p_1p_2, p_2p_3 and p_3p_4 are edges of P_4).

Let $U = V(G) \setminus N[P_4]$. If $U = \emptyset$, then $\gamma(G)$ is at most 4. Hence, the problem is in P. Now, we suppose that $U \neq \emptyset$. Clearly, U is an independent set because G is $2K_2$-free. Hence, $N(U) \subseteq N(P_4)$. As G is a connected graph, $N(u) \neq \emptyset$ for every $u \in U$. Moreover, if u, u' are two different vertices of U then $N(u) \cap N(u') = \emptyset$ by claw-freeness. Let v be any vertex in $N(U)$ which means v is neighbor of some $u \in U$. Clearly, $N(v) \cap P_4 = \{p_2, p_3\}$ because otherwise there is a $2K_2$ or a claw as induced subgraph which is a contradiction. Now, we will prove that $N(U)$ induces a complete subgraph. Suppose there are two different no adjacent vertices $v, v' \in N(U)$, then $\{p_1, p_2, v, v'\}$ induces a claw which is a contradiction. Hence, $N(U)$ induces a complete subgraph. We show that p_1 and p_4 cannot have a common neighbor. Suppose w is a common neighbor of p_1 and p_4. In this case, w belongs to $N(P_4)\setminus N[U]$. If w is adjacent to some vertex $v \in N(U)$, then $\{w, v, p_1, p_4\}$ induces a claw, a contradiction. Hence, w is not adjacent to any $v \in N(U)$. Choose any edge uv where $u \in U$. Clearly, $\{u, v, w, p_1\}$ induces a $2K_2$. Again, this is a contraction. In consequence, $N[p_1] \cap N[p_4] = \emptyset$ which implies that the closed neighborhoods of vertices in $U \cup \{p_1, p_4\}$ are all disjoint. Then, $\gamma(G) \geq |U| + 2$. Now, we prove that $D = U \cup \{p_2, p_3\}$ is a dominating set of G and it must be minimum. Suppose there is some vertex w is not dominated by D. Clearly, $w \in N(\{p_1, p_4\}) \setminus N[\{p_2, p_3\}]$. If w is not adjacent to p_1 (p_4), then $\{w, p_4, p_1, p_2\}$ ($\{w, p_1, p_3, p_4\}$) induces a $2K_2$ which is a contradiction. Hence, w is adjacent to p_1 and p_4. Again, this is a contradiction. Consequently, D is a (minimum) dominating set and it can be obtained in polynomial time. □

Theorem 2.2. *The domination problem restricted to* $(2K_2$,*diamond)-free graphs class is in* P.

Proof. Let G be a connected ($2K_2$,diamond)-free graph and $K_p = \{u_1, \ldots, u_p\}$ be a maximum clique in G, which can be obtained in polynomial time (the number of maximal cliques is polynomial). We separate the proof in two cases according to the size of K_p

- $p \leq 3$: Then G is $(2K_2, K_4)$-free. By Corollary 3.2 the problem is in P
- $p \geq 4$: If $G = K_p$ then $\gamma(G) = 1$. Otherwise, let $v \in N(K_p)$. It is easy to see that $|N(v) \cap K_p| = 1$, otherwise G is not diamond-free. Hence, if $S = V(G) \setminus K_p$ is an independent set, then $N(S)$ is a minimum dominating set. Otherwise, let $(v, w) \in E(G)$ such that $v, w \in S$. Suppose w.l.o.g. $u_1 \in N(v)$. Since G is $2K_2$-free then w must be connected to at least $p - 2$ vertices of $\{u_2, \ldots, u_p\}$ but then again G is not diamond-free, hence w cannot exist. Thus every vertex from $S = N(K_p)$ has degree one and $N(S)$ is a minimum dominating set.

□

Now we proceed to prove the complexity for the domination problem restricted to ($2K_2$,co-claw)-free is in P. We begin by showing a lemma that turns out to be useful for proving the desired property.

Lemma 2.9. *Let* G *be a connected* $(2K_2, K_3)$*-free graph with an induced* $C_5 = \{v_1, v_2, v_3, v_4, v_5\}$. *Then* C_5 *is a dominating set.*

Proof. Suppose that C_5 is not a dominating set and let w be a vertex from G which is not in $N[C_5]$. Since w cannot be an isolated vertex it must be connected with some other vertex $x \notin C_5$. Then $\{w, x\}$ forms a K_2. Since G is $2K_2$-free then (w, x) cannot be disjoint with any K_2 from C_5. In this case x must be connected to at least three vertices from C_5 which implies there is some K_3 formed by x and two consecutive vertices of C_5. Absurd since G is K_3-free. □

Theorem 2.3. *The domination problem restricted to* ($2K_2$,*co-claw)-free graphs class is in* P.

Proof. Let G be a connected ($2K_2$,co-claw)-free graph.

- If G is a tree then the problem is in P [87]. Hence it contains a cycle.
- If G contains a K_3 then let $V(K_3) = U \subseteq V(G)$. Thus any vertex from $V(G)$ must be in $N[U]$, otherwise U along with the non-adjacent vertex to U form a co-claw. Hence there is a dominating set of cardinal 3 and minimum dominating set can be obtained in $O(n^3)$. Therefore G is K_3-free.
- If G is C_5-free, and by prior items G is $2K_2$-free and K_3-free, then domination problem is in P since it is proved in [70] that the problem for $(2K_2, C_5, K_3)$-free graphs is in P.
- G is C_{n+5}-free ($n > 0$) because it is $2K_2$-free, and G contains a C_5 from previous item. We apply Lemma 3.2 described above and we affirm there is a dominating set of G with size at most 5. Thus the minimum dominating set can be found in $O(n^5)$.

□

Theorem 2.4. *The domination problem restricted to* ($2K_2$,*paw)-free graphs class is in* P.

Proof. Let $T = \{t_1, t_2, t_3\}$ be an induced K_3 from G. Let v be a vertex not adjacent to T and $P = \{u_1, \ldots, u_k = v\}$ a shortest path from T to v, w.l.o.g. $u_1 = t_1$. Then u_2 must be adjacent to t_1 and some other vertex of T, otherwise $\{u_2, t_1, t_2, t_3\}$ induces a paw. Without loss out generality, u_2 is adjacent to t_2. Let u_i be the vertex from P such that u_i is not adjacent to t_2 with smallest index i. Clearly, $\{t_2, u_{i-2}, u_{i-1}, u_i\}$ induces a paw which is a contradiction. If we take v as an arbitrary vertex from G, then we showed that any vertex is adjacent to T and the minimum dominating

set has at most three vertices. Thus the minimum dominating set can be found in $O(n^3)$. □

Note that K_3-free graphs are a subclass of Diamond-free graphs. Hence $(2K_2, K_3)$-free class is a subclass of ($2K_2$,Diamond)-free class. We know that dominating set problem restricted to ($2K_2$,Diamond)-free graphs is in P, then the problem is also in P for $(2K_2, K_3)$-free graphs.

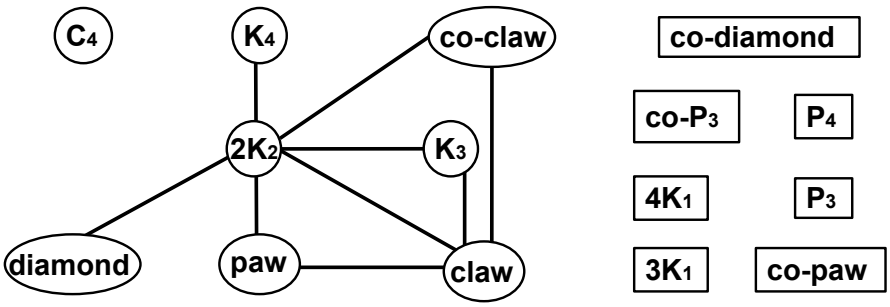

Figure 2.2: Scheme of graph classes complexity for the dominating set problem.

We will prove that Figure 2 is the final meta-graph, which represents the complexity of the minimum dominating set problem for any $\mathcal{F}$-free graphs, where $\mathcal{F}$ consists of graphs with at most 4 vertices.

Let W be the subset of the corresponding vertices in the meta-graph. Clearly, if W has a rectangular shape vertex or two vertices joined by an edge, the dominating problem is polynomial on $\mathcal{F}$-free graphs. It only remains to consider the case when W is an independent set of circular shape vertices. We will show that in this case, the problem is NPC.

Let W be an induced subgraph which is an independent set with only circular-shaped vertices. We can extend W to a maximal circular-shaped vertex-independent set W'. Clearly, W' corresponds to $\mathcal{F}'$-free graphs which is a subclass of $\mathcal{F}$-free graphs.

If we prove that the problem restricted to $\mathcal{F}'$-free graphs is NPC then the problem restricted to $\mathcal{F}$-free graphs is also NPC.

It is easy to see that there are exactly 3 maximal circular-shaped vertex-independent sets: $W_1 = \{C_4, 2K_2\}, W_2 = \{C_4, K_3, K_4$,diamond,paw,co-claw$\}$ and $W_3 = \{C_4, K_4$,diamond, claw$\}$. By [6], the problem restricted to $(C_4, 2K_2)$-free graphs is NPC. Theorem 2.5 proves the problem restricted to $(C_4, K_3, K_4$,diamond,paw,co-claw)-free graphs is NPC and Corollary 2.7 proves the problem restricted to $(C_4, K_4$,diamond, claw)-free graphs is NPC.

Lemma 2.10. *[79] If a graph G' is obtained from a graph G by triple subdivision of an edge, then $\gamma(G') = \gamma(G) + 1$.*

Theorem 2.5. *The domination problem restricted to (K_3, C_4)-free graphs is NPC.*

Proof. It is trivial to check that after applying a triple subdivision to every edge of an arbitrary graph $G = (V, E)$, the result graph G' is (K_3, C_4)-free. Applying Lemma 2.10, if we can solve in polynomial time the domination problem restricted to (K_3, C_4)-free graphs, then we can solve the problem on an arbitrary graph ($\gamma(G) = \gamma(G') + |E(G)|$). Consequently the domination problem restricted to (K_3, C_4)-free graphs is NPC. □

Definition 2.1. *Say a vertex v is a* claw-vertex *if $d(v) = 3$ and its neighbors form an independent set.*

Definition 2.2. *Let G be an arbitrary graph that has a claw-vertex v and its neighbors are w_1, w_2 and w_3. Say a* magnification *for v is the replacement of v by a cycle $C_9 = \{v_1, v_2, \ldots, v_9\}$ with three additional edges v_2v_9, v_3v_5 and v_6v_8, where v_1 connects to w_1,v_4 connects to w_2 and v_7 connects to w_3 (See Figure 3). We call the C_9 with three additional arcs H_9.*

Lemma 2.11. *Let G be a graph with a claw-vertex v. If G' is the resulting graph after magnification of v, then $\gamma(G') = \gamma(G) + 2$.*

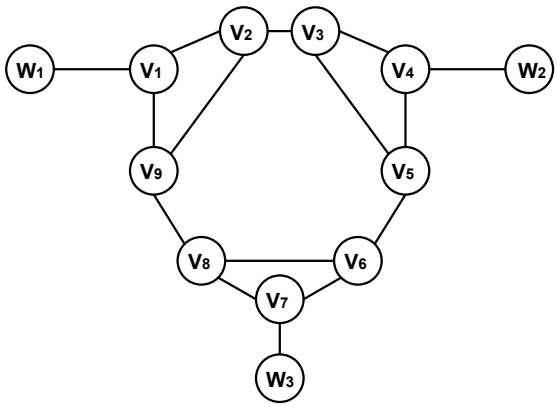

Figure 2.3: Replace each claw-vertex in the graph with this new structure,H_9.

Proof. Let D be a minimum dominating set of G, $|D| = \gamma(G)$. We construct a dominating set D' of G' in the following way:

1. $v \in D$: Thus $D' = (D \setminus v) \cup \{v_1, v_4, v_7\}$ is a dominating set of G' and $|D'| = |D| + 2$.

2. $v \notin D$: Suppose w.l.o.g. $w_1 \in D$. Then $D' = D \cup \{v_3, v_8\}$ is a dominating set of G' and $|D'| = |D| + 2$.

It is easy to check that in both cases D' is a dominating set of G' and $|D'| \leq |D|+2$. Then $\gamma(G') \leq |D'| \leq |D| + 2 = \gamma(G) + 2$.

Next, we will show that $\gamma(G') \geq \gamma(G) + 2$. Let D' a minimum dominating set of G', $|D'| = \gamma(G')$. As $N_{G'}[v_3] \cap N_{G'}[v_8] = \emptyset$ then $|D' \cap V(H_9)| \geq 2$. We analyze each possible value of $|\{w_1, w_2, w_3\} \cap D'|$. For each case, we construct a dominating set D of G such that $|D| \leq |D'| - 2$ which implies $\gamma(G') \geq \gamma(G) + 2$.

1. $|\{w_1, w_2, w_3\} \cap D'| = 0$: $N[v_1], N[v_4], N[v_7]$ are disjoint sets. Each neighborhood must contain a vertex from $D' \cap V(H_9)$. Hence, H_9 has at least 3 vertices from D'. We remove those vertices from D' and add v. The result set D is a dominating set of G.

2. $|\{w_1, w_2, w_3\} \cap D'| = 1$: Suppose w.l.o.g. $w_1 \in D'$ and $w_2, w_3 \notin D'$. In case D' contains at least three vertices from H_9, then we repeat the reasoning of the

previous item. Otherwise, there are exactly two vertices from H_9 that belong to D'. Those should be v_3 and v_8, hence w_2, w_3 are dominated with vertices outside H_9. Therefore, D can be obtained by removing $\{v_3, v_8\}$ from D'. Clearly, D is a dominating set for G.

3. $|\{w_1, w_2, w_3\} \cap D'| = 2$: Suppose w.l.o.g. $\{w_1, w_2\} \subseteq D'$. If in $|V(H_9) \cap D'| \geq 3$, we can apply the same reasoning as in the first item and obtain the same result. Otherwise, there are exactly two vertices from H_9 belong to D'. If $v_7 \in D'$, then no vertex from H_9 can dominate the set $\{v_9, v_2, v_3, v_5\}$. Thus, $v_7 \notin D'$, and w_3 is dominated by a vertex outside H_9. Therefore, the set $D = D' \setminus H_9$ is a dominating set of G.

4. $|\{w_1, w_2, w_3\} \cap D'| = 3$: $\{w_1, w_2, w_3\} \in D'$. Thus $D = D' \setminus H_9$ is a dominating set of G.

□

Theorem 2.6. *The domination problem restricted to maximum degree 3 planar $(K_4, C_4, C_5, C_7, C_8, C_9, C_{10}, C_{11}$,diamond,claw)-free graphs is NPC.*

Proof. The domination problem restricted to planar graph G of maximum degree 3 is NPC [57]. Given any planar graph G of maximum degree 3, we apply a triple subdivision for each edge of G and obtain a graph G'. It is easy to see that G' is a planar graph of maximum degree 3 and it is $(K_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}$, diamond)-free. Clearly, G and G' have the same number of claw-vertices. By Lemma 2.10, we know that $\gamma(G') = \gamma(G) + |E(G)|$. We can apply magnification for each claw-vertex of G' in order to remove claws, obtaining a graph G'' which is maximum degree 3 planar $(K_4, C_4, C_5, C_7, C_8, C_9, C_{10}, C_{11}$,diamond,claw)-free graph. Applying Lemma 2.11 we know that $\gamma(G'') = \gamma(G') + 2|U'| = \gamma(G) + |E(G)| + 2|U|$, where U' is the set of claw-vertices of G' and U is the set of claw-vertices of G. Therefore the problem remains in NPC restricted to maximum degree 3 planar $(K_4, C_4, C_5, C_7, C_8, C_9, C_{10}$, C_{11},diamond,claw)-free graphs. □

Corollary 2.7. *The domination problem restricted to* $(K_4, C_4$,*claw,diamond)-free graphs* G *is* NPC.

2.3 Results

With the given results it is easy to know to which complexity class belongs the domination problem when it is restricted by any combination of forbidden induced subgraphs of order three or four.

Forbidden Induced Subgraphs	Complexity	References
(K_3, C_4)-free	NPC	Theorem 2.5
$(K_4, C_4$,claw,diamond)-free	NPC	Theorem 2.7
$(2K_2, C_4)$-free	NPC	[6]
P_4	P	[32]
(co-paw)-free	P	[70]
co-diamond	P	Remark 2.2
$4K_1$	P	Remark 2.1
$(2K_2$,claw)-free	P	Theorem 2.1
(claw,paw)-free	P	[15, 38]
(claw,co-claw)-free	P	[15, 38]
$(2K_2$,paw)-free	P	Theorem 2.4
$(2K_2$,co-claw)-free	P	Theorem 2.3
$(2K_2$,diamond)-free	P	Theorem 2.2
$(2K_2, K_4)$-free	P	[105]

We close the gaps of missing information about the complexity of the minimum dominating set problem for graph classes given by forbidden induced subgraphs of order at most four. Extending this technique to graphs of order greater than four could lead to a long number of proofs, which may be not convenient, hence, the

number four seems to be the right limit when one should stop using a smart brute-force approach. Forbidden induced subgraphs of size at most four had been already studied on different problems [68], including a similar classification for clique-width [18]. Many of these classes define the first barrier where problems become P instead of NPC. On the other hand, it seems that the same technique can be applied to many of the classic problems in order to close the same gaps.

Chapter 3

Domination and Roman-Domination algorithms

The minimum Roman dominating function is a variant of the very well known minimum dominating set problem. Both problems remain NP-Complete when restricted to P_5-free graph class [6, 35], and bipartite graphs [6].

The minimum Roman dominating function problem problem was introduced as a variant of the minimum dominating set problem, and the motivation arise from an optimization problem in location of legions (ancient Roman army units) to protect the Roman Empire.

The idea is that different locations (vertices) may have at most two legions. A location where no legion is stationed is considered unsecured, but it can be protected by sending a legion from a nearby location (an adjacent vertex). Therefore, in order to protect the empire, the legions should be positioned in a way such that any location is either secured, or can be secured in short time. The problem of select the locations of the legions in order to use the minimum amount is exactly the minimum dominating set problem. Whenever some unsecured location was attacked or a rebellion occurs, a nearby legion can be sent to control the situation. However the movement of the legion leaves unsafe the position where the legion was stationed, and maybe other

nearby locations. Moreover, moving an army out of a location increases the chances of rebellions or attacks to that location, hence, those location cannot become unsecured. Therefore, an additional restriction given was that, no secured location should become unsecured after an attack on any place. This is equivalent to ask that each location is either secured, or has a nearby location with more than one legion. We refer to [35, 75] for more background on the historical importance and theoretical results for this problem.

We propose very simple and robust algorithms for determining the minimum dominating set and minimum Roman dominating function for arbitrary graphs which run in polynomial time whenever $\gamma(G)$ is a constant. The same algorithms are extended in order to solve these problems efficiently when restricted to P_4-free and $(P_5, (s,t)$-net$)$-free graphs. We give the definitions of these classes in the next section. Our algorithms improve previous known results for (P_5,bull)-free graphs [70] and (K_p, P_5)-free graphs (for fixed p) [105]. There are already linear-time algorithms to determine $\gamma(G)$ and $\gamma_{_R}(G)$ for any P_4-free graph G [32, 51, 75, 87]. To the best of our knowledge, all of them use some sophisticated structures such as cotrees, modular decompositions, homogeneous extensions, etc. or require obtaining an appropriate model from the original graph, and then applying the algorithm. The proposed algorithms are extremely simple and use the same core procedures, which makes them useful for practical purposes.

3.1 Preliminaries

An (s,t)-net graph is a split graph $G = (K \dot{\cup} I, E)$ where the complete set K is $\{u_1, \ldots, u_s\}$, the independent set I is $\{v_1, \ldots, v_t\}$, $t \leq s$ and $u_i v_j \in E(G)$ if and only if $i = j$.

A Roman dominating function of a graph $G = (V, E)$ is a function $f : V \to \{0, 1, 2\}$ such that every vertex x with $f(x) = 0$ is adjacent to at least one vertex y with $f(y) = 2$. Clearly, $V(G)$ is partitioned into three disjoint sets $V_0 = f^{-1}(0)$,

$V_1 = f^{-1}(1)$ and $V_2 = f^{-1}(2)$ by this function f. The weight of a Roman dominating function f is $f(V(G)) = \sum_{x \in V(G)} f(x) = |V_1| + 2|V_2|$. The minimum weight of a Roman dominating function of G is called the Roman domination number of G and is denoted by $\gamma_{_R}(G)$. The cardinality of a minimum dominating set of a graph G is denoted by $\gamma(G)$. It is known that $\gamma(G) \leq \gamma_{_R}(G) \leq 2\gamma(G)$ [75]. A *dominating* S for a graph G is a subgraph of G such that $N[S] = V(G)$. Without loss of generality, we assume that G is connected. Therefore, $n \in O(m)$.

Definition 3.1. *Let $W \subseteq V(G)$, define $F(W)$ the set of Roman domination functions f such that $f^{-1}(2) = W$.*

It is easy to see that the function $f \in F(W)$ such that $f^{-1}(0) = N(W) \setminus W$ and $f^{-1}(1) = V(G) \setminus N[W]$ minimizes Roman domination weight among functions in $F(W)$. We name this function as f_W. Hence $\gamma_{_R}(G) = \min_{W \subseteq V(G)} f_W(V(G))$.

Definition 3.2. *Let $g : V(G) \to \{0, 1\}$ where $g^{-1}(0) \subseteq N(g^{-1}(1))$ be a dominating function where the weight is defined as: $g(V(G)) = \sum_{v \in V(G)} g(v)$*

Let $\mathcal{D}$ be the set of all dominating functions and $\mathcal{R}$ the set of all Roman dominating functions. Therefore we can conclude:

- $\gamma(G) = \min_{g \in \mathcal{D}} g(V(G))$.
- Given a dominating function g, then $f = 2g$ is a Roman dominating function where $f^{-1}(1) = \emptyset$.
- Given a Roman dominating function f such that $f^{-1}(1) = \emptyset$ then $g = \frac{f}{2}$ is a dominating function.
- $\gamma(G) = \min_{g \in \mathcal{D}} g(V(G)) = \min_{f \in \mathcal{R} \ \wedge \ f^{-1}(1) = \emptyset} \frac{f(V(G))}{2} = \min_{W \subseteq V(G) \ \wedge \ f_W^{-1}(1) = \emptyset} \frac{f_W(V(G))}{2}$

3.2 Algorithms for general graphs

3.2.1 Domination

For the domination problem we propose a straightforward search algorithm consisting on looking up each subset H, and checking if there exists a vertex v such that $N[H \cup \{v\}] = V(G)$. Note that if such vertex v exists, then $\frac{f_{H\cup\{v\}}}{2}$ is a dominating function.

For this purpose, we define the procedure $FindBestAdditionalVertex$ that for any subset of vertices H finds a vertex v such that $|N[H \cup \{v\}]|$ is maximized. In order to achieve $O(m)$ time, the procedure should first mark vertices from the set $N[H]$, and then iterate through the neighbors of each candidate vertex (i.e. $V(G) \setminus H$) and maintain the vertex with most neighbors outside $N[H]$.

The pseudocode of the resulting algorithm is presented next.

Algorithm 1 GeneralDomination(Graph G)

```
for i ← 0 ... n − 1 do
  for each H: subset of i vertices do
    v ← FindBestAdditionalVertex(H, G)
    if N[H ∪ {v}] = V(G) then
      return H ∪ {v}
    end if
  end for
end for
```

Observe that the algorithm calls iteratively the described procedure $FindBestAdditionalVertex$ for every induced subgraph of G, named H, in increasing order according to its size. Thus when H is a subset of i vertices the algorithm would discover any dominating set of $i + 1$ vertices that includes H.

Since there are $O(n^i)$ subsets of size at most i, the algorithm running time is $O(n^{\gamma(G)-1}m)$.

3.2.2 Roman Domination

The problem can be solved using a simple modified version of $GeneralDomination(G)$. The idea is to use the generated sets H of vertices as the set V_2 of the Roman dominating function, therefore, exploring every possible Roman dominating function by making an exhaustive search for V_2. It is clear that for every H of different iteration of $GeneralDomination(G)$, there is a Roman dominating function f_{V_2} which is at least as good as any other Roman dominating function $f_W, H \subset W \wedge |W| = |H| + 1$ (where v is determined by $FindBestAdditionalVertex$ procedure). We keep the best f_{V_2} as f_Z during whole execution of $GeneralDomination(G)$. The algorithm stops when $|H| + 1 \geq \frac{f_Z(V(G))}{2}$. Clearly, $f_Z(V(G)) \leq f_W(V(G))$, for any $W \subseteq V(G)$ with $|W| < \frac{f_Z(V(G))}{2}$ because there is some $f_{W'}$ with $f_{W'}(V(G)) \leq f_W(V(G))$ and $|W'| = |W|$ which has been examined before. For any $W \subseteq V(G)$ with $|W| \geq \frac{f_Z(V(G))}{2}$, $f_W(V(G)) \geq 2|W| \geq f_Z(V(G))$. Therefore, f_Z is a minimum Roman dominating function and $\gamma_R(G) = f_Z(V(G))$.

The pseudocode of the resulting algorithm is presented next.

Algorithm 2 GeneralDomination(Graph G)

```
for i ← 0 ... n − 1 do
  if 2i ≥ best then
    return best
  end if
  for each H: subset of i vertices do
    v ← FindBestAdditionalVertex(H, G)
    best ← MIN{best, |V(G) \ N[H ∪ {v}]| + i}
  end for
end for
```

The running time is $O(n^{\lfloor \frac{\gamma_R(G)}{2} \rfloor - 1} m)$ because the number of H's to be considered is at most $O(n^{\lfloor \frac{\gamma_R(G)}{2} \rfloor - 1})$ (H has at most $\lfloor \frac{\gamma_R(G)}{2} \rfloor - 1$ vertices).

3.3 Algorithms for P_4-free graphs

In this section, we show an extremely simple linear time robust algorithm for both problems restricted to P_4-free graphs using the same approach. Suppose G is connected and $|V(G)| = n > 1$.

Lemma 3.1. *A non-trivial graph G is P_4-free if and only if $\exists v, w$ such that $N[v] \cup N[w] = V(G)$*

Proof. Let v be an arbitrary vertex from G. Assume G does not contain a vertex w such that $N[v] \cup N[w] = V(G)$. It is clear that if the distance of some pair of vertices $u, z \in V(G)$ is $k \geq 3$ then the shortest path connecting them is an induced P_{k+1}, a contradiction. Hence, every pair of vertices are at distance 1 or 2. The vertices in $U = V(G) \setminus N[v]$ are exactly those vertices at distance 2 from v. Clearly, $U \not\subset N(w)$, for each neighbor w of v, otherwise $N(v) \cup N(w) = V(G)$. Hence, there are $u_1, u_2 \in U$ and $w_1, w_2 \in N(v)$ such that $u_1 \in N(w_1) \setminus N(w_2)$ and $u_2 \in N(w_2) \setminus N(w_1)$. If $w_1 w_2 \in E(G)$ then $\{u_1, w_1, w_2, u_2\}$ induces a P_4, otherwise $\{u_1, w_1, v, w_2\}$ induces a P_4. In any case, we have a contradiction because the existence of an induced P_4. Such induced P_4 can be found in linear time and serves as a negative certificate.

□

The following algorithm follows from the lemma:

1. If the graph has an universal vertex v then: $\gamma(G) = 1$ ($\{v\}$ is a minimum dominating set) and $\gamma_{_R}(G) = 2$ ($f_{\{v\}}$ is a minimum Roman dominating function).

2. If there is vertex v such that $d(v) = n - 2$, and $w \notin N(v)$ then $\gamma(G) = 2$ and $\{v, w\}$ is a minimum dominating set.
   The Roman domination number $\gamma_{_R}(G)$ is 3 by defining: $f(v) = 2, f(w) = 1, f(N(v)) = 0$.

3. Choose an arbitrary vertex v and find $w \in N(v)$ such that $N(v) \cup N(w) = V(G)$. In affirmative case, $\gamma(G) = 2$ with $\{v, w\}$ as a minimum dominating set and $\gamma_{_R}(G) = 4$ with $f_{\{v,w\}}$ as a minimum Roman dominating function.

4. Otherwise, G is not P_4-free.

The total running time is $O(m)$. Note that Steps 1 and 3 of this algorithm can be done using two applications of procedure $FindBestAdditionalVertex$ employing $H = \emptyset$ for step 1 and $H = \{v\}$ for step 3.

3.4 Algorithms for $(P_5, (s,t)$-net$)$-free graphs

In this section, we will show algorithms for obtaining $\gamma(G)$ and $\gamma_{_R}(G)$ restricted to $(P_5, (s,t)$-net)-free graphs, where $t \leq s$. In case s is fixed, then the algorithm solves the problem in polynomial time.

The following lemmas of Chiba and Nishizeki [34] are helpful for our algorithms of this section.

Lemma 3.2. *[34] Given a graph G, $\sum_{uv \in E(G)} min\{d(u), d(v)\} \leq 2\alpha(G)m$ where arboricity $\alpha(G) \in O(\sqrt{m})$*

Lemma 3.3. *[34] Given a graph G, the number of K_p's of G is $O(\alpha(G)^{\frac{p-2}{2}}m)$ and can be list in $O(p \cdot \alpha(G)^{\frac{p-2}{2}}m)$ time.*

3.4.1 Domination

Let G be a connected P_5-free graph. Due to a result in [2], we know that G has either a dominating K_p or a dominating P_3.

Theorem 3.1. *[2] For each graph in the class of P_5-free graphs, there exists a dominating K_p, or a dominating P_3.*

Lemma 3.4. *If graph G has not a dominating P_3 and $\gamma(G) \geq 3$ then G does not contain a dominating C_4.*

Proof. Suppose there is a dominating $C_4 = \{u_1, u_2, u_3, u_4\}$. If this dominating set is not minimal, then there is a dominating P_3 or $\gamma(G) \leq 2$, a contradiction. Hence, it must be minimal and there are four vertices: $v_1 \in N(u_1) \setminus (N[u_2] \cup N[u_3] \cup N[u_4]), v_2 \in N(u_2) \setminus (N[u_3] \cup N[u_4] \cup N[u_1]), v_3 \in N(u_3) \setminus (N[u_4] \cup N[u_1] \cup N[u_2])$ and $v_4 \in N(u_4) \setminus (N[u_1] \cup N[u_2] \cup N[u_3])$. If $v_1v_2 \in E(G)$ then $\{v_1, v_2, u_2, u_3, u_4\}$ induces a P_5, a contradiction. Hence, $v_1v_2 \notin E(G)$. Using similar argument, $v_2v_3 \notin E(G)$. If $v_1v_3 \notin E(G)$ then $\{v_1, u_1, u_2, u_3, v_3\}$ induces a P_5, again this is a contradiction. Thus, $v_1v_3 \in E(G)$. But in this case, $\{v_1, v_3, u_3, u_2, v_2\}$ induces a P_5 which contradicts the P_5-freeness of G. Therefore, C_4 is not a dominating set. □

Theorem 3.2. *If graph G has not a dominating P_3 and $\gamma(G) \geq 3$ then the following conditions holds:*

(a) G has a minimal dominating K_p for some $p \geq 3$.

(b) For all minimal dominating K_p of G, G contains a (p, p)-net as induced subgraph.

Proof. (a) is a direct consequence of the fact that G has either a dominating K_p or a dominating P_3.
(b) Let $K_p = \{u_1, \ldots, u_p\}$ be a minimal dominating complete of G. Every vertex $v \notin D$ satisfies: $N(v) \cap D \neq \emptyset$. Since K_p is minimal, for each $i = \{1, \ldots, p\}$ there exists a vertex $v_i \in V(G) \setminus K_p$ such that $v_i \in N(u_i)$ and $v_i \notin N(u_j)$ for all $j \neq i$. Suppose $\exists\ v_iv_j \in E(G)$. Hence $C = \{u_i, u_j, v_j, v_i\}$ induces a C_4. By lemma 3.4, C is not a dominating set. There is some vertex $z \in V(G)$ such that $C \cap N(z) = \emptyset$. Thus, $\exists u_k \in D \setminus \{u_i, u_j\}$ such that $z \in N[u_k]$. In this case. $\{z, u_k, u_i, v_i, v_j\}$ induces a P_5, absurd. Therefore $v_iv_j \notin E(G)$ and $\{v_1, \ldots, v_p\}$ is an independent set which means $\{u_1, \ldots, u_p, v_1, \ldots, v_p\}$ induces a (p, p)-net. □

Corollary 3.1. *If G is $(P_5, (s, s)$-net$)$-free graph then $\gamma(G) \leq max\{3, s - 1\}$.*

Proof. If G contains a dominating P_3, then $\gamma(G) \leq 3 \leq max\{3, s-1\}$. Otherwise, by Theorem 3.2, G has a minimal dominating K_p (choose the largest one) and G has a (p,p)-net as induced subgraph. Since G is $((s,s)$-net)-free, $p \leq s-1$. Hence $\gamma(G) \leq p \leq s-1 \leq max\{3, s-1\}$. □

As a consequence of Corollary 3.1, there is a polynomial time algorithm to solve the minimum dominating set problem for $(P_5, (s,t)$-net)-free graphs. Next theorem gives an implementation of such algorithm based on procedure $FindBestAdditionalVertex$ described in Section 3.2.

Theorem 3.3. *For $(P_5, (s,t)$-net)-free graphs where s is a fixed value and $t \leq s$, the domination problem can be solved in*

- $O(m)$ *time, for* $s \leq 2$.
- $O(m^2)$ *time, for* $3 \leq s \leq 4$.
- $O(n^{s-3}m + m^{\frac{s}{2}})$ *time, for* $s \geq 5$.

Proof. If $s \leq 2$ then G is P_4-free. There are several linear time algorithms to solve domination problem for P_4-free graphs and we propose a robust linear time algorithm based on procedure $FindBestAdditionalVertex$ in next section.
For $s \geq 3$, applying procedure $FindBestAdditionalVertex$ with differents induced subgraph H we can find a minimum dominating set as follows:

1. Check if $\gamma(G) = 1$. $O(m)$. Using H as the empty set
2. Check if $\gamma(G) = 2$. $O(mn)$. Using H as each vertex.
3. Check if there is a P_3 dominating set (in the positive case, $\gamma(G) = 3$). $O(m^2)$. Using H as any edge.
4. By Theorem 3.2, there is a (p,p)-net as induced subgraph of G. Hence, $p \leq s-1$ and there is dominating K_p which implies that $\gamma(G) \leq s-1$.

- If $s \geq 5$, we check if minimum dominating sets have size at most $s-2$. This can be done in $O(n^{s-3}m)$ using all H with at most $s-3$ vertices.
- If $\gamma(G) = s-1$ then $p = s-1$. The dominating $K_p = K_{s-1}$ can be found in $O(m^{\frac{s}{2}})$ time using the procedure $FindBestAdditionalVertex$ for each induced K_{s-2} of G. The number of K_{s-2}'s is $O(m^{\frac{s-2}{2}})$ and can be list in $O((s-2)m^{\frac{s-2}{2}})$ time by Lemmas 3.2 and 3.3.

□

Corollary 3.2. *Dominating set problem can be solved in $O(m^2)$ in $(P_5, bull)$-free graphs.*

To the best of our knowledge the best algorithm for $(P_5, bull)$-free graphs has $O(n^6)$ time complexity [70].

Lemma 3.5. *All given algorithms are robust.*

Proof. From Theorem 3.3, in case there is not a P_5 nor an (s,t)-net in the graph, algorithms above find a dominating set. Otherwise the input graph has a forbidden structure. Returning a certificate is possible but the complexity of given algorithms may be increased by extra computations to find such certificate. Therefore, returning a certificate is not included in these algorithms. □

3.4.2 Roman Domination

Using Corollary 3.1 and the fact that $\gamma_{_R}(G) \leq 2\gamma(G)$, we have the following corollary.

Corollary 3.3. *If G is $(P_5, (s,s)$-net$)$-free graph then $\gamma_{_R}(G) \leq max\{6, 2s-2\}$.*

Theorem 3.4. *For $(P_5, (s,t)$-net$)$-free graphs where s is a fixed value and $t \leq s$, the Roman domination problem can be solved in*

- *$O(m)$ time, for $s \leq 2$.*

- $O(mn^2)$ *time, for* $s \leq 4$.
- $O(mn^{s-3} + m^{\frac{s}{2}})$ *time, for* $s \geq 5$.

Proof. If $s \leq 2$ then G is P_4-free. In [75], a linear time algorithm based on cotree is given to solve Roman domination problem for P_4-free graphs. Also, we describe a robust linear time algorithm to solve this problem in Section 3.3.
For $s \geq 3$, applying the following procedure we can find a minimum Roman dominating function as follows:

1. If $\gamma_{_R}(G) \leq 7$, then $\gamma_{_R}(G)$ can be determined in $O(mn^{\lfloor \frac{\gamma_R(G)}{2} \rfloor - 1})$ using the general algorithm described in subsection 3.2.2
2. In this case, $\gamma_{_R}(G) \geq 8$ implying $\gamma(G) \geq 4$. By Theorem 3.2, there is a (p, p)-net as induced subgraph of G. Hence, $p \leq s - 1$ and there is dominating K_p which implies that $\gamma(G) \leq s - 1$ and $\gamma_{_R}(G) \leq 2s - 2$.
   - Check if $\gamma_{_R}(G) \leq 2s - 3$. This can be done in $O(n^{s-3}m)$ using the general algorithm.
   - If $\gamma_{_R}(G) = 2s - 2$ then $\gamma(G) = s - 1$ and $p = s - 1$. The dominating $K_p = K_{s-1}$ can be found in $O(m^{\frac{s}{2}})$ time as we described in the proof of Theorem 3.3. Clearly, $f_{K_p}(V(G)) = 2s - 2$ and f_{K_p} is a minimum Roman dominating function.
   - If not such a minimum dominating set can be found then the input graph has a P_5 or (s, t)-net.

Clearly, the total complexity of the described algorithm is $O(mn^{s-3} + m^{\frac{s}{2}})$. For $s \leq 4$, by Corollary 3.3 $\gamma_{_R}(G) \leq 6$. Therefore, step 2 of above procedure can be omitted and the complexity is reduced to $O(mn^2)$. □

Similarly, the algorithms described in Theorem 3.4 are robust.

Corollary 3.4. *Roman Dominating problem can be solved in* $O(mn^2)$ *in* $(P_5, bull)$*-free graphs.*

Again, $(P_5, bull)$-free graphs are $(P_5, (3,2))$-net)-free. We can solve Roman domination problem using the $O(mn^2)$ algorithm which is faster than the best known $O(n^6)$ time algorithm [75] for this class of graphs.

Chapter 4

Efficient Edge Domination

Under the widely accepted assumption that $P \neq NP$ there are several problems with important applications for which no polynomial algorithm exists. The need to get an exact solution for many of those problems has lead to a growing interest in the area of design and analysis of exact exponential time algorithms for NP-Hard problems [55, 103]. Even a slight improvement of the base of the exponential running time may increase the size of the instances being tractable. There has been many new and promising advances in recent years towards this direction [8, 9].

In this chapter we give exact algorithms for the weighted and counting versions of the problem Dominating Induced Matching (also known as DIM or Efficient Edge Domination) which has been extensively studied [20, 22, 24, 27, 28, 29, 69, 80, 81], while we also developed efficient algorithms for the problem restricted to several graph classes. Further notes about this problem and some applications related to coding theory, network routing and resource allocation can be found in [61, 78].

First, we introduce the problem, giving its basic definitions and showing previous results. Then we describe three algorithms for the weighted and counting version of the problem. The first one runs in linear time, given a vertex dominating set of fixed size of the graph. The second one runs in polynomial time if the graph admits a polynomial number of maximal independent sets. The third one is an

$O^*(1.1939^n)$ time and polynomial (linear) space, which improves over the existing algorithms for exact solving this problem for general graphs. Then we show how to improve the EED algorithm restricted to chordal, dually-chordal, biconvex and claw-free graphs. We have also developed an efficient algorithm for circular-arc graphs, but we accommodate it in the next chapter for organizational purposes only, since some needed results will be given after this chapter.

4.1 Preliminaries

Given an edge $e \in E$, say that e *dominates* itself and every edge sharing a vertex with e. Subset $E' \subseteq E$ is an *induced matching* of G if each edge of G is dominated by at most one edge in E'. A *dominating induced matching (DIM)* of G is a subset of edges which is both dominating and an induced matching. Not every graph admits a DIM, and the problem of determining whether a graph admits it is also known in the literature as *efficient edge domination problem*. Given a function $\mathcal{E}(G) \to \mathcal{R}$, the weighted version of DIM problem is to find a DIM such that the sum of weights of its edges is minimum among all DIMs, if any. We denote the minimum weighted DIM problem for graph G with $DIM_\Omega(G)$ and the counting of DIMs in G as $DIM_C(G)$.

The minimum weighted DIM problem can be expressed as an instance of the maximum weighted independent set (MWIS) problem on the square of the line graph $L(G)$ of G, and also as an instance of the minimum weighted dominating set problem on $L(G)$, by slightly adjusting the models [22, 85] for the unweighted DIM problem. A detailed reduction of the weighted efficient domination problem to the MWIS problem is described in [19] We will use an alternative definition [29] of the problem of finding a dominating induced matching. It asks to determine if the vertex set of a graph G admits a partition into two disjoints subsets. The vertices of the first subset are called *white* and induce an independent set of the graph, while those of the second subset are named *black* and induce an 1-regular graph.

We assume the graph G to be connected, otherwise, the DIM of G is the union of the DIMs of its connected components.

For a coloring C of the vertices of G, denote by $C^{-1}(white)$ and $C^{-1}(black)$, the subsets of vertices colored white and black. A coloring C' is an *extension* of C if $C^{-1}(black) \subseteq C'^{-1}(black)$ and $C^{-1}(white) \subseteq C'^{-1}(white)$. For $V', V'' \subset V(G)$ if C' is obtained from C by adding to it the vertices of V' with the color black and those of V'' with the color white then write $C = C' \cup BLACK(V') \cup WHITE(V'')$.

4.2 Previous results

The unweighted version of the DIM problem is known to be NP-complete [61], even for planar bipartite graphs of maximum degree 3 [20] or k-regular graphs, for $k \geq 3$ [27]. There are polynomial time algorithms for some graph classes, such as chordal graphs [80], generalized series-parallel graphs [80] (both for the weighted problem), claw-free graphs [28], graphs with bounded clique-width [28], hole-free graphs [20], convex graphs [69], dually-chordal graphs [22], P_7-free graphs [24], bipartite permutation graphs [81], AT-free graphs [22], interval-filament graphs [22], weakly chordal graphs [22]. See also [23].

The MWIS problem can be solved in $O^*(1.2377^n)$ time [97] (how one obtains an algorithm for MWIS from an algorithm for weighted 2-Sat is described in [41]). On the other hand, the minimum weighted dominating set problem can be solved in time $O^*(1.5535^n)$ [56], and the special case where the weights are polynomially bounded in time $O^*(1.5048^n)$ [94]. Hence the minimum weighted DIM problem for a graph G can be solved by using the $L^2(G)$ algorithm in $O^*(1.2377^m)$ time using the MWIS algorithm and in $O^*(1.5048^m)$ time using the minimum weighted dominating set algorithm.

For the counting problem, there exist an algorithm [40] which can be used to count the number of MWIS's in $O^*(1.3247^n)$ time, leading to an $O^*(1.3247^m)$ time algorithm to count the numbers of DIMs.

A straightforward brute-force algorithm for obtaining the DIM of a graph G consists in finding all bipartitions of $V(G)$, coloring one of the parts as *white*, the other as *black*, and checking if the result is a valid DIM. The complexity of this algorithm is $O(2^n \cdot m) \in O^*(2^n)$.

It is not hard to see that every DIM is a maximum induced matching, and hence the number of edges of every DIM in G is the same. Therefore it is straightforward to modify the graph in order to solve the problem with non-negative weights and then transform it back to the original graph. The modification consists on adding the minimum weight of the graph to every edge. At the end, subtract it from the solution.

4.3 Efficient Edge Domination on general graphs

If $P \neq NP$ it is not possible to solve this problem in polynomial time, hence it becomes important to improve the exponential algorithm in order to identify instances that can be solved within reasonable time. Moreover, the result presented here is not only interesting from the theoretical point of view but also from a practical perspective since the algorithms presented have low polynomial and constants associated.

4.3.1 Colorings and Extensions

Assigning one of the two possible colors to vertices of G is called a coloring of G. A coloring is *partial* if only part of the vertices of G have colors assigned, otherwise it is *total*. A partial coloring is *valid* if no two white vertices are adjacent and no black vertex has more than one black neighbor. A black vertex is *single* if it has no black neighbors, otherwise, it is *paired*. A total coloring is *valid* if no two white vertices are adjacent and every black vertex is paired. Clearly, G admits a DIM if and only if it admits a total valid coloring. In fact, a total valid coloring defines exactly one DIM, given by the set B.

Next, we describe the natural conditions for determining if a coloring is valid or invalid.

Definition 1. *RULES FOR VALIDATING COLORINGS:*

A partial coloring is valid whenever:

(V1) *No two white vertices are adjacent, and*

(V2) *Each black vertex is either single or paired. Each single vertex has some uncolored neighbor.*

A total coloring is valid whenever:

(V3) *No two white vertices are adjacent, and*

(V4) *Each black vertex is paired.*

Lemma 4.1. *There is a one-to-one correspondence between total valid colorings and dominating induced matchings of a graph.*

Proof. It follows from the definitions. □

Definition 4.1. *We say $N_U(v)$ is the set of uncolored vertices from $N(v)$.*

Given a partial coloring C, the basic idea of the algorithm is to iteratively find extensions C' of C, until eventually a total valid coloring is reached. It follows from the validation rules that if C is invalid, so is C'. Therefore, the algorithm keeps checking for validation, and would discard an extension whenever it becomes invalid.

Basically, there are two different ways of possibly extending a coloring, using propagation rules and branching rules. At the beginning, there are partial colorings C which force the colors of some of the so far uncolored vertices, leading to an extension C' of C. In this case, say that C' has been obtained from C by *propagation*. The following is a convenient set of rules, whose application may extend C, in the above described way.

Lemma 4.2. *RULES FOR PROPAGATING COLORS:*

The following are forced colorings for the extensions of a partial coloring of G.

(P1) *The degree-3 vertices of a diamond must be black and the remaining ones must be white*

(P2) *The neighbor of a pendant vertex must be black*

(P3) *Each neighbor of a white vertex must be black*

(P4) *Except for its pair, the neighbors of a paired (black) vertex must be white*

(P5) *Each vertex with two black neighbors must be white*

(P6) *If a single black vertex has exactly one uncolored neighbor then this neighbor must be black*

(P7) *In an induced paw, the two odd-degree vertices must have different colors*

(P8) *In an induced C_4, adjacent vertices must have different colors*

(P9) *If $\forall v \in N_U(s), N[v] \subseteq N[s]$ where s is a single (black) vertex then an uncolored neighbor v of s minimizing weight(sv) must be black. Break ties arbitrarily. We require rules (P1) and (P8) to be applied before (P9).*

Proof. The rules (P1), (P7), (P8) follow from [20](using observations 1 and 3), while rules (P3), (P4), (P5), (P6) follow from [29]. The rule (P2) follows from the coloring definition since each black vertex must be paired in order for the coloring to be *valid*. Finally, for (P9), let s be a single vertex. Suppose the neighborhood of all uncolored neighbors of s lies within the neighborhood of s. Then the choice of the vertex to become the pair of s is independent of the choices for the remaining single vertices of the graph. Therefore, to obtain a minimum weighted dominating induced matching of G, the neighbor v of s minimizing $weight(sv)$ must be black. □

Lemma 4.3. *[20] If G contains a K_4 then G has no DIM.*

Say that a coloring C is *empty* if all vertices are uncolored. Let C be a valid coloring and C' an extension of it, obtained by the application of the propagation rules. If $C = C'$ then C is called *stable*. On the other hand, if $C \neq C'$ then C' is not necessarily valid. Therefore, after applying iteratively the propagation rules, we reach an extension which is either invalid or stable. In order to possibly extend a stable coloring C, we apply *branching rules*. Any coloring directly obtained by these rules is not forced. Instead, in each of the these rules, there are two possibly conflicting alternatives leading to distinct extensions C'_1, C'_2 of C. Each of C'_1 or C'_2 may be independently valid or invalid. The next lemma describes the branching rules. We remark that there exist simpler branching rules. However, using the rules below we obtain a sufficient number of vertices that get forced colorings, through the propagation which follow the application of any branching rule, so as to guarantee a decrease of the overall complexity of the algorithm. The complexity obtained relies heavily on this fact.

In general, we adopt the following notation. If C is a stable coloring then S denotes the set of single vertices of it , U is the set of uncolored vertices and $T = U \backslash \cup_{s \in S} N_U(s)$.

Lemma 4.4. *BRANCHING RULES*

Let C be a partial (valid) stable coloring of a graph G. At least one of the following alternatives can be applied to define extensions C'_1, C'_2 of C.

(B1) *If C is an empty coloring: choose an arbitrary vertex v then $C'_1 := C \cup BLACK(\{v\})$ and $C'_2 := C \cup WHITE(\{v\})$*

(B2) *If $\exists$ edge vw s.t. $v \in N_U(s)$ and $w \in N_U(s')$, for some $s, s' \in S, s \neq s'$ then $C'_1 := C \cup BLACK(\{v\})$ and $C'_2 := C \cup WHITE(\{v\})$*
(Please see Figure 4.1)

(B3) *For some $s \in S$, if $\exists v \in N_U(s)$ s.t. $\exists w \in N_T(v)$:*

B3(a) *If* $|N_U(s)| \neq 3 \vee d(w) \neq 3 \vee |N_T(v)| \geq 2$ *then* $C'_1 := C \cup BLACK(\{v\})$ *and* $C'_2 := C \cup WHITE(\{v\})$.

B3(b) *If* $|N_U(s)| = 3 \wedge d(w) = 3 \wedge N_T(v) = \{w\}$, *let* $N_U(s) = \{v, v', v''\}$.

B3(b).i *If* $N_U(v') = N_U(v'') = \emptyset$ *then* $C'_1 := C \cup BLACK(\{v\})$ *and* $C'_2 := C \cup WHITE(\{v\})$

B3(b).ii *If* $N_U(v') \neq \emptyset$, *let* $w' \in N_T(v')$, *with* $w' \neq w$. *If* $|N(w) \cup N(w')| > 5$ *or* $ww' \notin E(G)$ *then* $C'_1 := C \cup BLACK(\{v\})$ *and* $C'_2 := C \cup WHITE(\{v\})$

B3(b).iii *If* $N_U(v') \neq \emptyset$, *let* $w' \in N_T(v')$, *with* $w' \neq w$. *If* $ww' \in E(G)$ *and* $z \in N(w) \cap N(w')$ *then* $C'_1 := C \cup BLACK(\{v''\})$, *while if* $weight(sv) + weight(w'z) \leq weight(sv') + weight(wz)$ *then* $C'_2 := C \cup BLACK(\{v\})$, *otherwise* $C'_2 := C \cup BLACK(\{v'\})$
(Please see Figure 4.2)

Proof. If C is an *empty* coloring then rule (B1) applies. Otherwise, if C is not an *empty* coloring and C is not a *total* coloring we will show that $S \neq \emptyset$. Since C is not *total* and the graph is connected, there is at least one edge sv where v is uncolored. If s is white then v must be black (P3), else if s is a *paired* vertex then v must be white (P4). Therefore s must be a single black vertex, hence $S \neq \emptyset$. Let $s \in S$. Since C is valid then $N_U(s) \neq \emptyset$ by (V2) and since is *stable* $|N_U(s)| \neq 1$ by (P6). Therefore $|N_U(s)| \geq 2$. Moreover rule (P9) cannot be applied, therefore $\exists v \in N_U(s)$ s.t. $|N_U(v) \setminus N(s)| > 0$, let $w \in N_U(v) \setminus N(s)$. If $\exists s' \in S, s \neq s'$ s.t. $w \in N_U(s')$ then rule (B2) is applied. Otherwise, suppose rule (P2) cannot be applied. Then $w \in N_T(v)(|N_T(v)| \geq 1)$. Clearly, $d(w) \neq 1$, otherwise, rule (P2) must be applied and v must get color black.

In case $|N_U(s)| \neq 3$ or $d(w) \neq 3$ or $|N_T(v)| \geq 2$ we apply rule B3(a). Otherwise: $|N_U(s)| = 3, d(w) = 3, |N_T(v)| = 1$. Note that in B3(b), $v'w'$ behaves symmetrically in respect to vw since otherwise $v'w'$ were found in step B3(a) replacing vw.

The first subcase of B3(b) corresponds to $N_U(v') = N_U(v'') = \emptyset$, while in the second and third subcases, v' or v'' has uncolored neighbors. Note that if $v'v'' \in E(G)$ then either a paw, a diamond or a K_4 is induced by $svv'v''$, therefore v is a colored vertex and this case never occurs.

Suppose w.l.o.g. $N_U(v') \neq \emptyset$ where $w' \in N_T(v')$. It is easy to see that $w \neq w'$ since otherwise $svwv'$ is a C_4 and therefore w cannot be uncolored by rule (P8).

Now there are three cases which lead to two possible outcomes from the algorithm: In case $ww' \in E(G)$ or $|N_U(v) \cup N_U(w)| > 5$ then the result of the algorithm is given by the second subcase (ii), else it is given by the third subcase (iii). Note that $\{v, v', v''\} \in N_U(s)$ while $\{w, w'\} \in T$, hence these vertices are different since they belong to disjoint sets. Also note that $\exists\ z \in N(w) \cap N(w')$ since otherwise the connected component has 7 vertices and can be trivially solved. □

Each rule is applied after the previous rule, that is, if the condition of the previous case is not verified in the entire graph. Note that this applies to subitems of case (B3).

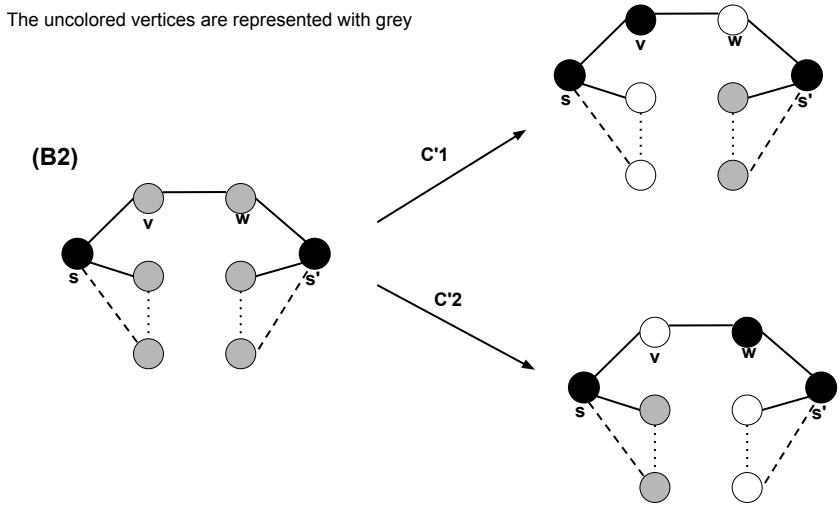

Figure 4.1: Branching rule sample - (B2)

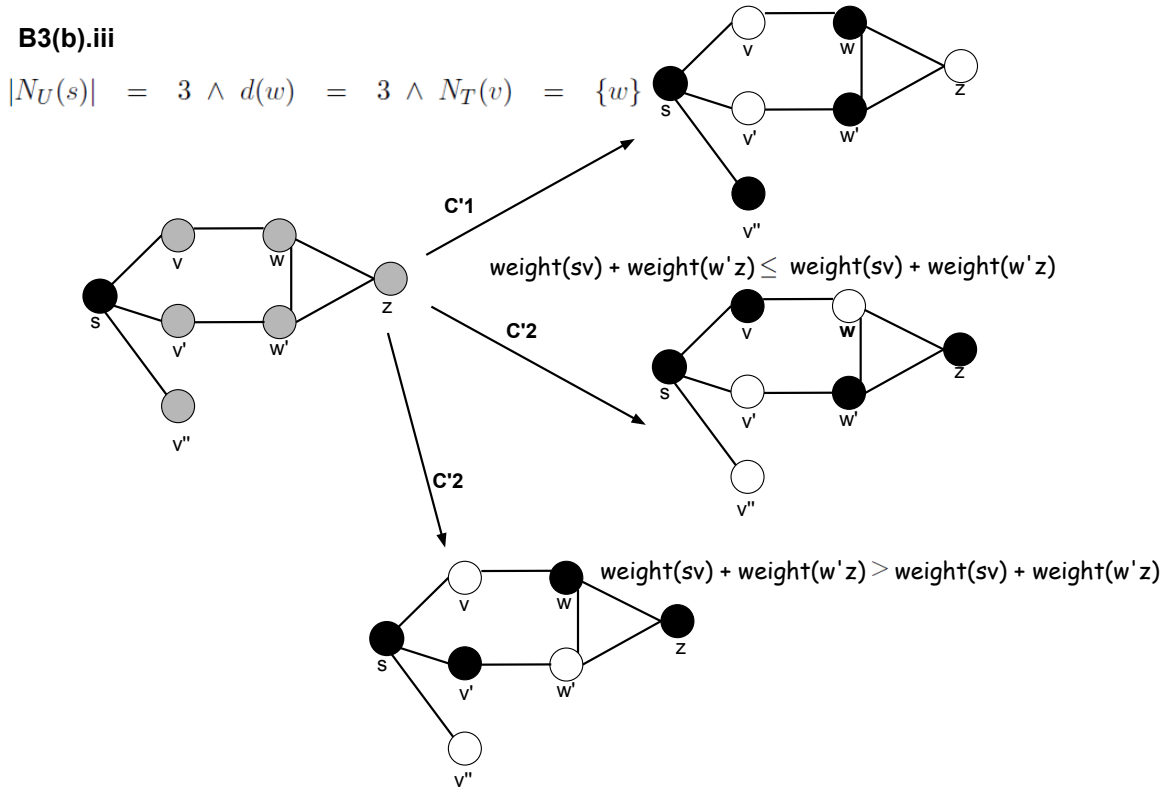

Figure 4.2: Branching rule sample - B3(b).iii

4.3.2 An algorithm based on vertex domination

We propose an exact algorithm for solving the weighted dominating induced matching problem, for general graphs, based on vertex dominations. The only rules to be used will be the propagation ones.

Let C be a *partial* valid coloring of $G = (V, E)$. Such as in [29], this coloring can be further propagated according to the previous defined propagation rules:

Let C be a partial valid coloring of G, and C' be a stable coloring obtained from C, by the applications of rules (P1)-(P9). Denote by D and D', respectively the subsets of vertices of G which are colored in C and C'. Clearly, $D' \supseteq D$. For our purposes, assume that the initial set D of colored vertices is a vertex dominating set of G.

Lemma 4.5. *Let C' be a stable coloring. Then*

1. *If there are no single (black) vertices then C' is a total coloring,*
2. *Any uncolored vertex has exactly one black neighbor, and such a neighbor must be single.*

Proof. Recall that D are the initial colored vertices and is a dominating set of the graph G. C' is not a total coloring if only if there is some uncolored vertex v. Clearly, $v \notin D$ and $N(v) \cap D \neq \emptyset$. Let w be some neighbor of v in D. If the color of w is white then v must be colored black by rule (P3) which is a contradiction. Hence w is a black vertex. Again, if w is a paired black vertex or v has another black neighbor $w' \neq w$, v must get color white by rules (P4) and (P5) and this is a contradiction. Consequently, v has exactly one black neighbor and which is single black vertex. Therefore, if there are no single black vertices then there are no uncolored vertices and C' is a total coloring. □

Let D' be the colored vertices of the stable coloring C', let $S = \{s_1, \ldots, s_p\}$ be the set of single vertices, and U the set of still uncolored vertices of G, that is, $U = V \setminus D'$. The above lemma implies that U admits a partition into (disjoint) parts:

$$U = (N(s_1) \cap U) \cup \ldots \cup (N(s_p) \cap U)$$

Theorem 4.1. *Let C be a coloring of the vertices of G, C' a stable extension of it, and $D = C^{-1}(black) \cup C^{-1}(white)$ a dominating set of G. Then (i) $S \subseteq C^{-1}(black)$; and (ii) if C extends to a valid total coloring C'' then C'' is an extension of C'.*

Proof. Suppose that $S \not\subseteq C^{-1}(black)$ which means that exists a vertex $s_i \in S$ and $s_i \notin C^{-1}(black)$. By definition of S, s_i is a single black vertex. If $s_i \in D$ then $s_i \in C^{-1}(black)$, contradiction. Therefore $s_i \notin D$.

If D is a dominating set, then $\exists v \in D$ such that $s_i \in N(v)$. If v is black then s_i is not a single black vertex, again a contradiction. Hence v must be white.

- If s_i has no uncolored neighbors then C is not extensible to a total valid coloring because s_i cannot become a paired vertex, contrary to the hypothesis.

- Otherwise, let y be an uncolored neighbor of s_i. Clearly, $y \notin D$. Since y is uncolored then it has exactly one neighbor in D'. That is, s_i is the unique neighbor of y in D'. Since $D \subseteq D'$ and $s_i \in D' \setminus D$, it follows that D is not a dominating set, contradiction.

On the other hand, C' and C'' are extensions of C. Then the vertices of D have the same color in these colorings. Any colored vertex $v \notin D$ of C' was obtained by some propagation rule base on previous colored vertices. The rules are correct and deterministic. Hence, v must have the same color in C'' and C'' is an extension of C'. □

Clearly, given a partial valid coloring C, we can compute efficiently a stable extension C' of it. In addition, if D is a dominating set then we can try to obtain a total valid coloring from the stable coloring C' by appropriately choosing exactly one vertex from each subset $N_U(s_i)$, to be black, that is, to be the pair of the so far single vertex s_i.

Lemma 4.6. *Let U and S, respectively be the sets of uncolored and single vertices, relative to some stable coloring C' of graph G. If C' extends to a total valid coloring then, for each $s_i \in S$, $G[N_U(s_i)]$ is a union of a star and an independent set, any of them possibly empty. Moreover, the pair of s_i must be a maximum degree vertex in $G[N_U(s_i)]$.*

Proof. Suppose by contrary that $G[N_U(s_i)]$ is not a union of a star and an independent set. Then $G[N_U(s_i)]$ contains either two non-adjacent edges, or a K_3.

- Let $\{(u_1, u_1), (v_1, v_2)\}$ be two disjoint edges in $G[N_U(s_i)]$. Since no white vertices can be adjacent, let u be the black vertex from $\{u_1, u_2\}$ and v the black vertex from $\{v_1, v_2\}$. Then $\{u, s_i, v\}$ is a black P_3 or K_3 and therefore cannot be extended to a valid coloring.

- Let $\{(u_1, u_2, u_3)\}$ be a K_3 in $G[N_U(s_i)]$. Therefore $\{s_i, u_1, u_2, u_3\}$ is a K_4 and therefore G has no valid coloring.

Consequently, $G[N_U(s_i)]$ must be a union of a star and an independent set. Now, suppose by contrary that the pair of s_i is a vertex $v \in N_U(s_i)$ and v has not maximum degree in $G[N_U(s_i)]$. Clearly, the rest of vertices in $N_U(s_i)$ are white vertices. In particular, a maximum degree vertex u in $G[N_U(s_i)]$ is white. But, there is some neighbor $z \neq v$ of u in $N_U(s_i)$ and z is not adjacent to v. Hence, z and u are white adjacent vertices, which is a contradiction. □

We can repeatedly execute the procedure below described for choosing the vertices to be paired to the single vertices s_i of the partial colorings. The procedure is repeated until all parts of the partition $U = N_U(s_1) \cup \ldots \cup N_U(s_p)$ have selected their paired black vertices or the coloring becomes invalid.

Let $s_i \in S$ be a single vertex. *Case 1*: $N_U(s_i) = \emptyset$: then stop, it will not lead to a valid one. *Case 2*: There is exactly one maximum degree vertex in $G[N_U(s_i)]$: then clearly, the only alternative is to choose this vertex. *Case 3*: There is no edge vw, where $v \in N_U(s_i)$ and $w \in N_U(s_j)$, for any $j \neq i$: then the choice of the neighbor of s_i to become black is independent on the choices of the others parts of the partition. Choose the vertex w of maximum degree in $G[N_U(s_i)]$ that minimizes the weight of the edge ws_i. *Case 4*: There is an edge vw, where $v \in N_U(s_i)$ and $w \in N_U(s_j)$, for some $j \neq i$: then v may become white if and only if w may become black. Each of these two choices may lead to valid or invalid total colorings. So, we proceed with both alternatives, as if in parallel.

After applying any of the above Cases 2, 3 or 4, perform the propagation rules again and validate the coloring so far obtained. Proceed so until eventually the coloring becomes invalid, or a valid solution is obtained. At the end, choose the minimum weight solution obtained throughout the process.

As for the complexity, it is clear that it depends on the cardinality of the dominating set D and on the number of parallel iterations, considered sequentially. Next, we describe bounds for these parameters.

Lemma 4.7. *There are at most 2^q parallel computations where $q \leq p = |S| \leq |D|$, and $q \leq \frac{n}{3}$.*

Proof. : By Theorem 4.1, it follows that $p \leq |D|$. On the other hand, we can apply the above Cases 1-4, in such an ordering that we keep applying Cases 1 and 2, with propagation until all remaining single vertices s_i satisfy $|N(s_i) \cap U| \geq 2$. Let $S' \subset S$ denote the set of remaining single vertices, and $q = |S'|$. Consequently, $q \leq \frac{n}{3}$.

Next, examine the parallel computations. They are generated by Case 4. Let vw be an edge of G, where $v \in N(s_i) \cap U$ and $w \in N(s_j) \cap U$, $i \neq j$. In one of the instances, v is black, meaning that s_i becomes paired, while in the other one w is black, implying that s_j becomes paired. This means that the size of the set S' of single vertices always decreases by at least one unit in all computations. Hence there are at most 2^q parallel computations. □

Considering that the remaining operations involved in each parallel thread of the algorithm can be performed in linear time, it is not hard to conclude that there there is an $O(2^q m)$ time algorithm to obtain a minimum DIM, if any, extensible from a partial valid coloring C of a weighted graph $G = (V, E)$ such that $D = C^{-1}(black) \cup C^{-1}(white)$ is a dominating set of G.

The complexity of the algorithm depends on the size of the dominating set D employed. We remark that if $G = (V, E)$ has no isolated vertices then we can easily find in linear time a dominating set with at most half the vertices. Just determine a maximal independent set I. Clearly, I and $V \setminus I$ are both dominating sets of G and one of them has at most $\frac{n}{2}$ vertices.

Finally, in order to obtain the minimum weighted DIM of the graph G, we have to apply the described algorithm for all possible bi-colorings of D. There are exactly $2^{|D|}$ such colorings. Therefore

Theorem 4.2. *There is an algorithm of complexity* $O(min\{2^{2|D|}, 2^{\frac{5n}{6}}\} \cdot m) = O^*(min\{4^{|D|}, 1.7818^n\})$ *to compute a minimum weighted DIM of a weighted graph, if existing.*

Proof. The complexity is $O(2^{|D|} \cdot 2^q \cdot m) = O(2^{|D|} \cdot 2^{min\{|D|, \frac{n}{3}\}} \cdot m) = O(2^{|D|} \cdot min\{2^{|D|}, 2^{\frac{n}{3}}\} \cdot m) = O(min\{2^{2|D|}, 2^{|D|+\frac{n}{3}}\} \cdot m) = O(min\{2^{2|D|}, 2^{\frac{n}{2}+\frac{n}{3}}\} \cdot m) = O(min\{2^{2|D|}, 2^{\frac{5n}{6}}\} \cdot m)$. □

Corollary 4.1. *The above algorithm solves the minimum weighted DIM problem in* $O(m)$ *time given a dominating set of fixed size.*

4.3.3 An algorithm based on maximal independent sets

In this section, we describe an exact algorithm for finding a minimal weighted DIM of a graph, based on enumerating maximal independent sets. We consider a weighted graph $G = (V, E)$.

Any maximal independent set $I \subseteq V$ induces a partial bi-coloring in G as follows:

- color as black all vertices of $V \setminus I$
- color as white the vertices of I except those having exactly one single neighbor.

Observation 1. *If all vertices of* G *have degree* $\neq 1$ *then the above partial coloring is total.*

The algorithm is then based on the following lemma.

Lemma 4.8. *Let* G *be a graph,* I *a maximal independent set of it and* C *the partial bi-coloring induced by* I*. Then* C *is extensible to a DIM if and only if* C *is a valid coloring and each single vertex, if existing, has at least one uncolored neighbor in* C*.*

Proof. $\Rightarrow$) It is easy to see that if C is not a valid coloring, then it is not extensible to a DIM. Besides, if C has a single vertex v with no uncolored neighbors then all neighbors of v are white in C and in any extension of C. Also,C is not extensible to

a DIM because v cannot ever get its pair.

$\Leftarrow$) Let C be a valid coloring where each single black vertex has at least one uncolored neighbor. Then for each single black vertex v, choose any uncolored neighbor w to be its pair (w has exactly one single neighbor) and the remaining of uncolored vertices get color white. In this total coloring, the black vertices induce an 1-regular subgraph and the white vertex set is an independent set because it is part of I. Hence, the total coloring is valid and hence a DIM. □

The algorithm can then be formulated as follows. Generate the maximal independent sets I of G. For each I, find its induced coloring C. If C is invalid or some single vertex has no uncolored neighbor then do nothing. Otherwise, for each single vertex v in C, if any, choose the minimum weight vw, among the uncolored neighbors of v; then color w as black and the remaining neighbors of v as white. The set of black vertices then forms a DIM of G. At the end select the minimum weight among all DIMs obtained in the process, if any.

Clearly, this algorithm determines the minimum weight DIM of a weighted graph $G = (V, E)$ because given any DIM $E' \subseteq E$ of G, the vertex set formed by those vertices not incident to any of the edges of E' is an independent set and as such, is clearly a subset of some maximal independent set of G. So, any DIM E' is considered in the algorithm.

All the operations performed by the algorithm relative to a fixed maximal independent set can be performed in linear time $O(m)$. If G has μ maximal independent sets, we can generate them all in time $O(nm\mu)$ time [93]. Therefore the complexity of the entire algorithm is $O(nm^2\mu)$. On the other hand, $\mu \leq O(3^{\frac{n}{3}})$, leading to a worst case of $O(3^{\frac{n}{3}}nm^2) \approx O^*(1.44225^n)$ time. In particular, if G is a bipartite graph then $\mu \leq 2^{\frac{n}{2}}$ and the complexity reduces to $O^*(1.41421^n)$. In any case, if G has a polynomial number of maximal independent sets then the algorithm terminates within polynomial time.

Finally, we observe the following additional relation between maximal independent sets and DIMs.

Lemma 4.9. *Let $G(V, E)$ be a graph with no isolated edges, $E' \subseteq E$ a DIM of G, and $I \subseteq V$ the independent set formed by those vertices not incident to any of the edges of E'. Then I is contained in exactly one maximal independent set of G.*

Proof. : If I is a maximal independent set there is nothing to prove. Otherwise, suppose the lemma is false and let I_1, I_2 be two distinct maximal independent sets properly containing I. Let $V_1 = I_1 \setminus I$, and $V_2 = I_2 \setminus I$. Choose any $v_2 \in V_2$. Clearly, $\{v_2\} \cup I$ is an indepedent set, and we know that $I_1 = V_1 \cup I$ is a maximal one. Consequently, there must be some vertex $v_1 \in V_1$ adjacent to v_2. However, both v_1 and v_2 are vertices incident to edges of the DIM E'. Consequently, $v_1v_2 \in E'$. In this case, v_1v_2 must form an isolated edge of G, a contradiction. Therefore the lemma holds. □

Based on the above lemma and that fact that every isolated edge must be part of any DIM, it is simple to extend the exact algorithm proposed in this section, so as to count the number of distinct DIMs (unweighted or minimum weighted) of G, in the same complexity as deciding whether G admits a DIM. Observe that G may contain an exponential number of DIMs.

4.3.4 An $O^*(1.1939^n)$ algorithm for $DIM_\Omega(G)$ and $DIM_C(G)$

Here we propose an algorithm for solving the problems of finding the minimum weighted DIM and that of counting the DIMs in $O(m \cdot 1.1939^n) \in O^*(1.1939^n)$ time and $O(m)$ space in general graphs, which improves over the existing algorithms. We employ techniques described in [55] for the analysis of our algorithm, and as such we use their terminology. The proposed algorithm was designed using the *branch & reduce paradigm*. More information about this design technique as well as the running time analysis for this kind of algorithms can be found in [55].

The algorithm

The lemmas described in the coloring section 4.3.1 lead to an exact algorithm for finding a minimum weight DIM of a graph G, if any.

In the initial step of the algorithm, we find the set $K4$ containing the K_4's of G. If $K4 \neq \emptyset$, by Lemma 4.10, G does not have DIMs, and terminate the algorithm. Otherwise, define the set $COLORINGS$ to contain through the process the candidate colorings to be examined and eventually extended.

Next, include in COLORINGS an *empty* coloring. In the general step, we choose any coloring C from $COLORINGS$ and remove it from this set. Then iteratively propagate the coloring by Lemma 4.2 into an extension C' of it, and validate the extension by Definition **1**. The iterations are repeated until one of the following situations is reached: C' is invalid, C' is a total valid coloring, or a partial stable (valid) coloring. In the first alternative, C' is discarded and a new coloring from $COLORINGS$ is chosen. If C' is a a total valid coloring, find its weight and if smaller than the least weight so far obtained, it becomes the current candidate for the minimum weight of a DIM of G. Finally, when C' is stable we extended it by branching rules: choose the first rule of Lemma 4.4 satisfying C', compute the extensions C' and C'', insert them in $COLORINGS$, select a new coloring from $COLORINGS$ and repeat the process.

The formulation below describes the details. The propagation and validation of a coloring C are performed by the procedure $PROPAGATE - VALIDATE(C, RESULT)$. At the end, the returned coloring corresponds to the extension C' of C, after iteratively applying propagation. The variable RESULT indicates the outcome of the validation analysis. If C' is invalid then $RESULT$ is 'invalid'; if C' is a valid total coloring then it contains 'total', and otherwise $RESULT$ equals 'partial'.

Algorithm Minimum Weighted DIM / Counting DIM

1. Find the subset $K4$

if $K4 \neq \emptyset$ **then** terminate the algorithm: G has no DIM

$SOLUTION := NO - DIM$

2. $COLORINGS := \{C\}$, where C is an *empty* coloring

3. **while** $COLORINGS \neq \emptyset$ **do**

a. Choose $C \in COLORINGS$ and remove it from $COLORINGS$

b. $PROPAGATE - VALIDATE(C, RESULT)$

c. **if** $RESULT =$ 'total' **and** $weight(C) < SOLUTION$ **then**

$SOLUTION := weight(C)$

else if $RESULT =$ 'partial' **then**

Set C_1' and C_2' according to branching RULES on C

$COLORINGS := COLORINGS \cup \{C_1', C_2'\}$

end if

4. **Output** $SOLUTION$

procedure $PROPAGATE - VALIDATE(C, RESULT)$

Comment Phase 1: Propagation

1. $C' := C$

2. repeat

$C := C'$

C' := extension of C obtained by the PROPAGATION RULES

until $C = C'$

Comment Phase 2: Validation

3. Using the VALIDATION RULES **1** do as follows:

if C is an invalid coloring **then return** (C, 'invalid')

else if C is a partial coloring **then return** (C,'partial')

else return (C, 'total')

Correctness and Complexity

It is easy to see that our algorithm fits the *branch* & *reduce* paradigm [55]. The *propagation rules* can be mapped into *reduction rules*.

Theorem 4.3. *The algorithm described in the previous section correctly computes the minimum weight of a dominating induced matching of a graph G.*

Proof: The correctness of the algorithm follows from the fact that the algorithm considers all colorings that represent a DIM that can have minimum weight. Lemmas 4.2 and 4.4 are applied to extend partial colorings. Invalid colorings are discarded, while valid colorings are further extended, except if some other valid coloring representing a better DIM (with less weight) appeared before.

For proving the worst-case running time upper bound for the algorithm we will use the following helpful definition and theorem.

Definition 2. *[55] Let b be a branching rule and n the size of the instance. Suppose rule b branches the current instance into $r \geq 2$ instances of sizes respectively at most $n-t_1, n-t_2, \ldots, n-t_r$, for all instances of size $n \geq max\{t_i : i = 1, 2, \ldots, r\}$. Then we call $b = (t_1, t_2, \ldots, t_r)$ the branching vector of branching rule b. The branching vector $b = (t_1, t_2, \ldots, t_r)$ implies the linear recurrence:*

$$T(n) \leq T(n - t_1) + T(n - t_2) + \ldots, T(n - t_r).$$

Theorem 4.4. *[55] Let b be a branching rule corresponding to the branching vector $(t_1, t_2, \ldots, t_r)$. Then the running time of the branching algorithm using only branching rule b is $O^*(\alpha^n)$, where α is the unique positive real root of*

$$x^n - x^{n-t_1} - x^{n-t_2} - \ldots - x^{n-t_r} = 0$$

The unique positive real root α is the *branching factor* of the branching vector b. We denote the branching factor of $(t_1, t_2, \ldots, t_r)$ by $\tau(t_1, t_2, \ldots, t_r)$.

Therefore for analyzing the running time of a branching algorithm we can compute the factor α_i for every branch rule b_i, and an upper bound of the running time of the branching algorithm is obtained by taking $\alpha = max_i \alpha_i$ and the result is an upper bound for the running time of $O^*(\alpha^n)$.

The upper bound is obtained by counting the leaves of the search tree given by the algorithm, using the fact that each leaf can be executed in polynomial time. The complexity of the algorithm without hiding the polynomial factor depends on the time for the execution of each branch in the search tree.

Further notes on this topic can be found in [55]

Theorem 4.5. *The algorithm above described requires $O^*(1.1939^n)$ time and $O(n+m)$ space for completion.*

Proof. Using Definition 4.3.4 and Theorem 4.4 the computation of the upper bound time is reduced to calculating the *branching vector* for each branching rule (i.e. branching rules in our algorithm) and obtaining the associated *branching factor* for

each case. Then the bound is given by the maximum *branching factor*. Note that it is required that the *reduction rules* (i.e. propagation rules in our algorithm) can be computed in polynomial time and leads to at most one valid extension of the considered coloring. So, the propagation rules do not affect the exponential factor of the algorithm. Moreover, each branch of the algorithm has cost $O(n+m)$ in time and space. This is easy to note since from the *empty* coloring up to any *total* coloring each vertex v is colored once and the cost for coloring each vertex is given by the updating of the color of the vertex and its neighbors, hence $O(|N(v)|)$ time. Therefore, the total cost for each branch is $O(n+m)$.

We consider the size of an instance (a coloring) is the number of uncolored vertices after application of propagation (reduction) rules. Let's analyze each branching rule to obtain the maximum *branching factor*:

1. If C is an *empty* coloring: choose an arbitrary vertex v then $C'_1 := C \cup BLACK(\{v\})$ and $C'_2 := C \cup WHITE(\{v\})$: It is easy to see that this rule is executed once, after that, the coloring is never empty again. Since each application of a branching rule originate two branches, we can bound the time of the algorithm by twice the complexity of for computing an instance of size $n-1$. Therefore the asymptotic behavior of the algorithm is not affected.

2. *If* $\exists$ *edge* vw *s.t.* $v \in N_U(s)$ *and* $w \in N_U(s')$*, for some* $s, s' \in S, s \neq s'$ *then* $C'_1 := C \cup BLACK(\{v\})$ *and* $C'_2 := C \cup WHITE(\{v\})$.

Here we extend the original coloring C' to C'_1 and C'_2 by coloring the vertex v with black and white respectively. Recall that there exists an edge vw such that $v \in N_U(s)$, $w \in N_U(s')$. If v is black then $N_U(s) \setminus v$ are white, while w is white. On the other hand, if v is white then w is black and $N_U(s') \setminus w$ are white. Therefore the size of uncolored vertices is reduced for each branch (i.e. for each new coloring). The associated branching vector is $(1 + |N_U(s)|, 1 + |N_U(s')|)$. By rule (P2) $|N_U(s)| \geq 2$ and $|N_U(s')| \geq 2$. The following observation turns out to be useful:

$\forall s_i \in S$ If $|N_U(s_i)| = 2$ then $N_U(s_i)$ can be totally colored, whether v is black or white. Therefore the branching vector with biggest branching factor is (3,5) ($\tau(3,5) \approx 1.1939$) and occurs whenever one of $\{s, s'\}$ has two uncolored neighbors and the other one has three uncolored neighbors.

3. For some $s \in S$, if $\exists v \in N_U(s)$ s.t. $\exists w \in N_T(v)$:

Note that if $\nexists w \in N_T(v)$ for any $v \in N_U(s)$ then either the propagating rule (P9) or (P5) can be applied to get an extension of the coloring.

(a) If $|N_U(s)| \neq 3 \vee d(w) \neq 3 \vee |N_T(v)| \geq 2$ then $C'_1 := C \cup BLACK(\{v\})$ and $C'_2 := C \cup WHITE(\{v\})$.

If v is uncolored then w is not a *pendant* vertex, $d(w) > 1$. If w is uncolored then it has neither a white nor a paired black neighbor. Moreover, if w has a single black neighbor then this is the case analyzed above. Therefore w has uncolored neighbors and let x be one of them.

(a.1) $|N_T(v)| \geq 2$: Let $v' \in N_T(v)$. Using the same reasoning that with w, we claim: $\exists x' \in N_U(v')$. In C'_1, $\{v, x\}$ will be black, while $\{x, v', w\}$ will be white In C'_2, $\{v\}$ will be white while $\{v', w\}$ will be black. This lead to the branching vector (3,5).

(a.2) $d(w) \neq 3$. If $d(w) = 2$ then in C'_1 the vertices $N_U(s) \cup \{w, x\}$ will be colored and in C'_2 the vertices $\{v, x\}$ will be black, while $\{w\}$ will be white. Therefore the branching vector with biggest branching factor is (3,5).

Else if $d(w) > 3$ then in C'_1 the vertices $N_U(s) \cup N_U[w]$ will be colored and in C'_2 the vertices $\{v, w\}$ will be colored. In case $|N_U(s)| = 2$ then v_1 will be colored too. Therefore the branching vectors are either

(2,7) ($\tau(2,7) = 1.1908$) or (3,6) ($\tau(3,6) = 1.1739$).

(a.3) $|N_U(s)| = 2$: Let $N_U(s) = \{v, v_1\}$ and $N(w) = \{v, x, x'\}$. In C'_1 after applying propagation rules the vertices $\{v, x, x'\}$ will be black, while $\{v_1, w\}$ will be white. In C'_2 after applying propagation rules the vertices $\{v_1, w\}$ will be black, while $\{v\}$ will be white. The result is the branching vector (3,5).

(a.4) $|N_U(s)| > 3$: Let $\{v_1, v_2, v_3\} \subseteq N_U(s)$ and $N(w) = \{v, x, x'\}$. In C'_1 after applying propagation rules the vertices $\{v, x, x'\}$ will be black, while $\{v_1, v_2, v_3, w\}$ will be white. In C'_2 after applying propagation rules the vertices $\{w\}$ will be black, while $\{v\}$ will be white. The result is the branching vector (2,7)

(b) *If* $|N_U(s)| = 3 \ \wedge \ d(w) = 3$ *where* $N_U(w) = \{v, x, x'\}, N_U(s) = \{v, v', v''\}$ Note that $\{x, x'\} \cap \{v, v', v''\} = \emptyset$ since otherwise at least one of them must be colored by rule (P8).

(b.1) *If* $N_U(v') = N_U(v'') = \emptyset$ *then*
$C'_1 := C \cup BLACK(\{v\})$ *and* $C'_2 := C \cup WHITE(\{v\})$:

Suppose w.l.o.g. $weight(sv') \leq weight(sv'')$, then:
In C'_1, after applying the propagation rules the vertices $\{v, x, x'\}$ will be black, while, $\{v', v'', w\}$ will be white.
In C'_2, after applying propagation rules the vertices $\{v', w\}$ will be black, while $\{v, v''\}$ will be white. The result is the branching vector (4,6) ($\tau(4,6) = 1.1510$).

(b.2) *If* $N_U(v') \neq \emptyset$, *let* $w' \in N_T(v')$, *with* $w' \neq w$*:*

If $|N_T[w] \cup N_T[w']| > 5$ then

$C'_1 := C \cup BLACK(\{v\})$ and $C'_2 := C \cup WHITE(\{v\})$

Note that if $d(w') \neq 3$ then $v'w'$ satisfies the properties of an already analyzed case, hence $C'_1 := C \cup BLACK(\{v'\})$ and $C'_2 := C \cup WHITE(\{v'\})$.

If $d(w) = d(w') = 3$ and $|N_T[w] \cup N_T[w']| > 5$, then $\exists x, y$ s.t. $x \in N_T(w), x \notin N_T(w')$ and $y \in N_T(w'), and \notin N_T(w)$. In C'_1 after applying propagation rules the vertices $\{v, x, x', w'\}$ will be black, while $\{v', v'', w\}$ will be white. If $x' = w'$ then y must be black by rule (P6). In C'_2 the vertex $\{w\}$ will be black, while the vertex $\{v\}$ will be white. The result is the branching vector (2,7)

(b.3) *If* $N_U(v') \neq \emptyset$, *let* $w' \in N_T(v')$, $w' \neq w$

If $|N_T[w] \cup N_T[w']| \leq 3$ *and* $z \in N(w) \cap N(w')$ then

$C'_1 := C \cup BLACK(\{v''\})$,

if $weight(sv) + weight(w'z) \leq weight(sv') + weight(wz)$ then

$C'_2 := C \cup BLACK(\{v\})$

otherwise $C'_2 := C \cup BLACK(\{v'\})$

If $d(w) = d(w') = 3$ then $ww' \in E(G)$ and $\exists z \in N_T(v) \cap N_T(w)$, otherwise the case is one of the above.

In both colorings, C'_1 and C'_2 the vertices $\{v, v', v'', w, w', z\}$ will be colored. The branching vector is (6,6) ($\tau(6,6) = 1.1225$).

The worst branching factor is $\tau(3,5) \approx 1.1939$. In consequence, the time complexity of this algorithm is $O^*(1.1939^n)$. To achieve linear space complexity, we use a stack to store the coloring sequence of the current branch.

□

Counting the number of DIMs

The previous algorithm can be easily adapted to count the number of DIMs. Given a coloring C we define $TVC(C)$ the number of *total valid* colorings that can be extended from C. *If we apply any propagation rule to coloring C we obtain a coloring C'. Clearly $TVC(C) = TVC(C')$, except for rule (P9). In the latter case $TVC(C) = TVC(C') \cdot |N_U(s)|$ where s is the single vertex chosen to apply the rule.*

Note that by swapping s for any vertex $v \in N_U(s)$ we get another valid DIM, since it satisfy Definition 1. from valid coloring. Thus, each vertex $v \in N_U(s)$ defines a different DIM.

If we apply any branching rule to coloring C we obtain two extended colorings C'_1 and C'_2. Clearly $TVC(C) = TVC(C'_1) + TVC(C'_2)$, except for rule B3(b).iii. In the latter case $TVC(C) = TVC(C'_1) + 2 \cdot TVC(C'_2)$.

Note that since C'_1 and C'_2 have a different color for at least one vertex, therefore the colorings from $TVC(C'_1)$ are disjoint from the colorings of $TVC(C'_2)$ and we must count each one separately. On the other hand, the colorings from $TVC(C'_2)$ can have either v black or v' black, each one leading to a different DIM. Thus, each coloring of those vertices gives a different coloring to be counted. Therefore there are $2 \cdot TVC(C'_2)$ different DIMs.

Using the above facts it is trivial to modify the algorithm to solve the counting problem.

We have shown algorithms that solve the problem for general graphs with exponential running time, however their running time is polynomial under certain conditions, such as having a polynomial number of maximal independent sets, or a dominating set of fixed size. For the last algorithm, if no more than a fixed number of branches are needed, the running time is polynomial, however it is not easy to identify the graphs where this happens.

4.4 Efficient Edge Domination on several restricted graph classes

In this section we describe $O(n)$ time algorithms for finding the minimum weighted dominating induced matching of chordal, dually chordal, biconvex, and claw-free graphs. For the first three classes, we prove tight $O(n)$ bounds on the maximum number of edges that a graph having a dominating induced matching may contain. By applying these bounds, and employing existing $O(n+m)$ time algorithms we show that they can be reduced to $O(n)$ time. For claw-free graphs, we describe a variation of the existing algorithm for solving the unweighted version of the problem, which decreases its complexity from $O(n^2)$ to $O(n)$, while additionally solving the weighted version. We show how the same algorithm can be easily modified to count the number of DIMs of the given graph.

4.4.1 Chordal, Dually Chordal and Biconvex graphs

We remark that computing $DIM_\Omega(G)$ for any graph G which is chordal, dually chordal or biconvex requires no more than $O(n)$ time.

Lemma 4.10. *[20] If G contains a K_4 then G has no DIMs.*

Lemma 4.11. *Every K_4-free chordal graph G with at least 2 vertices has at most $2n-3$ edges. The bound is tight even if G is an interval graph.*

Proof. By induction on the number of vertices of the graph. For $n = 2$, the result follows since such a graph has at most one edge. Suppose the bound is valid for any graph with $n-1$ vertices, $n \geq 3$. Let G be an n-vertex chordal graph and v a simplicial vertex of it. Since $|E(G)| = |E(G \setminus \{v\})| + d(v)$, by the induction hypothesis, the number of edges of $G \setminus \{v\}$ is bounded by $2(n-1) - 3 = 2n - 5$. Since G is K_4-free, $d(v) \leq 2$, therefore $|E(G)| \leq 2n - 5 + 2 = 2n - 3$.

An interval graph having two universal vertices and the remaining ones having degree 2 has no K_4 and contains $2n-3$ edges, meaning that the bound is tight for interval graphs. □

Corollary 4.2. *The $DIM_\Omega(G)$ problem can be solved in $O(n)$ time for (dually) chordal graphs.*

Proof. Let G be a given chordal graph. First, we count the number of edges of G, up to a limit of $2n-3$. If the bound has been exceeded then stop answering that G has no DIMs. Otherwise, apply the algorithm [80] which will solve $DIM_\Omega(G)$ in $O(n)$ time. Finally, if a graph has a DIM then it is chordal if and only if it is dually chordal [22]. Consequently, $DIM_\Omega(G)$ can also be solved in $O(n)$ time for dually chordal graphs. □

Next, we consider solving $DIM_\Omega(G)$ for biconvex graphs. An ordering $<$ of X in a bipartite graph $G=(X,Y,E)$ has the *interval property* if for every vertex $y \in Y$, the vertices of $N(y)$ are consecutive in the ordering $<$ of X. A bipartite graph (X,Y,E) is *convex* if there is an ordering of X or Y that fulfills the interval property. Furthermore if there are orderings for both X and Y which fulfill the interval property the graph is *biconvex*.

Lemma 4.12. *Let G be a convex bipartite graph having no subgraph isomorphic to $K_{3,3}$. Then G contains at most $2n-4$ edges, for $n \geq 3$.*

Proof. The proof is by induction on n. If $n=3$ then it is trivial to verify that G satisfies the bound since it has at most 2 edges. Let G be an arbitrary $K_{3,3}$–free convex graph, v its minimum degree vertex and G' the graph obtained from G by removing v.

- $d(v) \leq 2$: Clearly, G' is also $K_{3,3}$–free. By inductive hypothesis, G' has at most $2(n-1)-4 = 2n-6$ edges. Consequently, G has at most $2n-6+d(v) \leq 2n-4$ edges.

- $d(v) > 2$: Every vertex in G has degree at least 3. Let $G = (X, Y, E)$ where X has the interval property. Thus for each vertex $y \in Y, N(y)$ consists of vertices that are consecutive. Let $\{x_1, \ldots, x_k\}$ be the ordering $<$ of X and w.l.o.g. let $\{y_1, y_2, y_3\} \subseteq N(x_1)$. Since y_1, y_2, y_3 have at least 3 neighbors and X has the interval property, it follows that $\{x_2, x_3\} \subseteq N(y_1) \cap N(y_2) \cap N(y_3)$. Therefore $\{x_1, x_2, x_3, y_1, y_2, y_3\}$ induces a $K_{3,3}$, which is a contradiction.

Hence, G contains indeed at most $2n - 4$ edges. This bound is tight, $K_{2,n-2}$ is an example. □

We remark that bipartite graphs, not necessarily convex, which do not contain $K_{3,3}$ as a minor also have at most $2n - 4$ edges [33]. However, this bound does not apply to general bipartite graphs not containing $K_{3,3}$ as an induced subgraph, as shown by the example below described.

Let $G = (X, Y, E)$ be a bipartite graph where $X = \{x_0, x_1, \ldots, x_{15}\}$ and $Y = \{y_0, y_1, \ldots, y_7\}$. Add the edge $x_i y_{2j}$, if the binary representation of i has the digit 0 at position j, while if such a binary representation contains the digit 1 at j then add the edge $x_i y_{2j+1}$. It is easy to see that one of the edges $x_i y_{2j}, x_i y_{2j+1}$ will exist for all $i, j : 0 \leq i \leq 15, 0 \leq j \leq 3$. We show that G is $K_{3,3}$-free: Suppose this is not true, and let $\{x_i, x_j, x_k, y_p, y_q, y_r\}$ be the vertices of an induced $K_{3,3}$. By the construction of G the binary representations of i, j, k have the same value for positions $\lfloor \frac{p}{2} \rfloor, \lfloor \frac{q}{2} \rfloor, \lfloor \frac{r}{2} \rfloor$. But i, j, k are distinct integers $0 \leq i, j, k \leq 15$, which leads to a contradiction, since there are no three integers smaller than 16 with the above property. Consequently, G is $K_{3,3}$–free. To complete the example, note that G has 24 vertices and more than 44 edges.

Give k copies of the graph defined above. Say x_j^i is the x_j vertex from the *i-th* copy. Add edges $y_7^i x_0^{(i+1)}, 0 \leq i < k$. The number of vertices is $24k$ while $m = 64k + k - 1$. This result graph is $K_{3,3}$-free bipartite and has all vertices of degree at least 4. The bound $65k - 1 \leq 48k - 4$ is not satisfied for any k.

Lemma 4.13. *Let $G = (X, Y, E)$ be a biconvex graph which has a DIM. Then G is $K_{3,3}$–free.*

Proof. Suppose G contains a $K_{3,3}$ given by $X' = \{x_1, x_2, x_3\} \subseteq X$ and $Y' = \{y_1, y_2, y_3\} \subseteq Y$. Consider an arbitrary DIM of the graph and its corresponding black-white coloring of the vertices. Then the vertices of X' and Y' must have distinct colors. Suppose w.l.o.g. that the vertices X' are black and those of Y' are white. Let y_1^*, y_2^*, y_3^* be the black neighbors of x_1, x_2, x_3, respectively. It follows that the graph induced by the nine vertices of $X' \cup Y' \cup \{y_1^*, y_2^*, y_3^*\}$ is not biconvex, a contradiction. □

Corollary 4.3. *The DIM problem for biconvex graphs can be solved in $O(n)$ time.*

Proof. Let G be a biconvex graph. If G contains a DIM, by Lemma 4.13, G is $K_{3,3}$–free. Therefore G has at most $2n-4$ edges, by Lemma 4.12. Consequently, given an arbitrary biconvex graph, count its number of edges, up to $2n-4$. If the number of edges exceeds $2n-4$ then the graph does not contains any DIM, otherwise apply the algorithm [20], which solves the DIM problem in $O(n+m)$ time, for chordal bipartite graphs. Since convex graphs are contained in chordal bipartite, we can solve the DIM problem for biconvex graphs in $O(n)$ time. □

We remark that there are convex graphs having a quadratic number of edges that admit DIMs. For instance, $V(G) = V_1 \cup V_2 \cup V_3$, where $|V(G)| = n, |V_1| = |V_2| = |V_3| = \frac{n}{3}$. Let V_i be an independent set for $1 \leq i \leq 3$, and let $V_1 \cup V_2$ induce a complete bipartite graph, $V_1 \cup V_3$ be an induced matching, and $V_2 \cup V_3$ be an independent set. Such a graph is bipartite, with bipartition $(V_1, V_2 \cup V_3)$, moreover it is convex bipartite since it admits a interval ordering. Also, it contains a quadratic number of edges. On the other hand, $V_1 \cup V_3$ is a DIM of it.

4.4.2 Claw-free graphs

The problem of finding a DIM on a claw-free graph, if existing, has been solved in [28] by an $O(n^2)$ time algorithm. We review the ideas of this paper and propose an improvement of it.

We assume that the given graph $G = (V, E)$ is connected, and is neither an induced cycle nor an induced path. Clearly, if G is disconnected we can reduce the problem to its connected components, while if G is a cycle or a path the solution is trivial.

By [28], if a claw-free graph G has a DIM then each vertex v of G is one of the following six types:

(1) degree 1

(2) degree 2 with two non-adjacent neighbors;

(3) degree 2 with two adjacent neighbors

(4) degree 3 with $G[N(v)]$ inducing a $K_1 + K_2$

(5) degree 3 with a $G[N(v)]$ inducing a P_3

(6) degree 4 with $G[N(v)]$ inducing a $2K_2$

Thus, we assume that each vertex of G falls into one of the above types. This implies $m \leq 2n$, i.e. $m = O(n)$.
In particular, the two edges incident to a Type 4 vertex v, which are contained in a triangle of $G(N[v])$, are called *heavy*, while the third edge incident to v is a *light* one. The algorithm [28] can be viewed as a sequence of the following distinct phases:

1. Handling three consecutive vertices of Type 2

2. Handling vertices of Type 1 which are at distance at least 3 of some Type 4 vertex

3. Coloring all vertices of Types 1,2,5 and 6

4. Coloring the remaining vertices, of Types 3 and 4

Our proposed algorithm describes new formulations for Phases 1,2 and 4, while maintaining the original Phase 3 of the algorithm [28]. We proceed by describing each of the parts.

Phase 1

The purpose is to eliminate the occurrence of three consecutive Type 2 vertices v_1, v_2, v_3, such that $N(v_2) = \{v_1, v_3\}$, $N(v_1) = \{v_2, w_1\}$ and $N(v_3) = \{v_2, w_3\}$. Consider the following alternatives:

- $w_1 = w_3$: In this case if $d(w_1) = 2$ then $G = C_4$, which contradicts G not to be a cycle. Hence $d(w_1) \geq 3$, but then $G[N[w_1]]$ contains a claw, a contradiction. Thus this case does not occur.

- $w_1w_3 \in E(G)$: If $d(w_1) = d(w_3) = 2$ then $G = C_5$ again a contradiction. Hence we may suppose $\exists u \in N(w_1) \setminus \{v_1, w_3\}$. We know that $u \notin N(v_1)$, thus in order to avoid a claw in $G[N[w_1]]$ we must assume $u \in N(w_3)$. The latter implies that no more vertices can belong to the neighborhoods of w_1 and w_3, otherwise G would contain vertices outside the above six types, a contradiction.

  Any DIM of G must have exactly one edge of the triangle $\{w_1, u, w_3\}$. The edge w_1w_3 does not lead to a valid DIM since it forces v_2 to be a single black vertex without black neighbor. It is easy to verify that the possibilities are either: $\{w_1u, v_2v_3\}$ or $\{w_3u, v_1v_2\}$.

  Therefore we can eliminate vertices v_1, v_2, v_3 and sum the weight of edge v_1v_2 to that of w_3u, and sum the weight of v_2v_3 to that of w_1u. To guarantee that the edge w_1w_3 is not chosen to enter the DIM, we assign infinite weight to it.

- $w_1 \neq w_3$ and $w_1w_3 \notin E(G)$: In this case we use the original procedure of [28], which consists of replacing vertices v_1, v_2, v_3 for the edge w_1w_3. However, the algorithm [28] solves the DIM problem without weights, thus, in order to

guarantee the correct solution for the new weighted graph, we need to consider the following additional possibilities:

- w_1, w_3 are black: Then v_1, v_3 are black and v_2 is white. The weights of edges v_1w_1 and v_3w_3 must be added to the weight of w_1w_3
- w_1 is black and w_3 is white: In this case, v_2 and v_3 are black while v_1 is white. Hence the weight of edge v_2v_3 must be added to the weight of each edge of the set of edges w_1z, where $z \neq v_1, z \neq w_3$
- w_3 is black and w_1 is white: This case is symmetric to the previous one. The weight of edge v_1v_2 must be added to the weight of each edge of the set w_3z, where $z \neq v_3, z \neq w_1$.

Note that these are disjoint changes, and we must apply all of them. These modifications to the original graph G are repeated until no three consecutive vertices of Type 2 remain in the graph, leaving a new reduced graph $G' = (V(G'), E(G'))$. This can be achieved in $O(n)$ time. The algorithm now proceeds on G'.

Phase 2

In this phase, we eliminate the occurrence of Type 1 vertices, lying at distance at least 3 from some Type 4 vertex. Let $v \in V(G')$ such that $d(v) = 1$ and let $w \in V(G')$ be the vertex such that $d(w) \geq 3$ and the distance to v is minimum. Note that if there is no such w then G' is a path, a contradiction. Therefore there is a path $v - w$ where all vertices, except v, w are of Type 2. Since there are at most two consecutive vertices of Type 2, the distance between v and w is at most 3. It is easy to see that w is of Type 4, otherwise G' is not claw-free. Let v, u_1, u_2, w be any path of length 3 from a vertex $v \in V(G')$ to a vertex $w \in V(G')$, with $d(v) = 1$ and $d(w) = 3$. Let $\{z_1, z_2\}$ be the unique K_2 in the subgraph induced by $N(w)$, and G^* be the graph after deletion of vertices $\{v, u_1, u_2\}$. It is clear that any DIM M^* of G^* contains exactly one edge from the triangle formed by wz_1z_2. In case M^* contains the edge

z_1z_2, we add the edge u_1u_2 to M^* in order to obtain a DIM of G, hence to generate a DIM with the same weight in G^* we set $\omega(z_1z_2) = \omega(z_1z_2) + \omega(u_1u_2)$. In case that M^* contains wz_1 or wz_2 the edge vu_1 is added to M^*. In the latter situation, we set $\omega(wz_1) = \omega(wz_1) + \omega(vu_1)$ and $\omega(wz_2) = \omega(wz_2) + \omega(vu_1)$. We repeat this process for each vertex $v \in V(G')$ such that $d(v) = 1$. Finally, we assert that every vertex of Type 1 is at distance 1 or 2 from some vertex of Type 4. These computations can be completed in $O(n)$ time.

Phase 3

By applying convenient propagation rules, the algorithm [28] colors a subset of vertices of the graph, including all vertices of Types 1,2,5, and 6. Let Γ be the final coloring so obtained in the algorithm. First, check its validity. If Γ is not valid, then G has no DIMs and the algorithm terminates. If C is valid and total, also terminate the algorithm, since the unique DIM of G has been found. Otherwise, proceed to Phase 4.

All the above operations can be completed in $O(n)$ time. At the end of this phase, the only possibly uncolored vertices are of Types 3 and 4. Observe that the obtained coloring is unique, so there is no choice to be made concerning weights, so far.

Phase 4

In this phase, we extend the coloring Γ, obtained by the previous phase, into a total valid coloring. It is assumed that Γ is a valid not total coloring, which cannot be extended by propagation. Let U be the set of uncolored vertices and S the set of single black vertices of the coloring Γ. Note that extending Γ is equivalent to extending the coloring Γ' of $G^*[U \cup S]$ (in Γ', only vertices of S are colored with black color). It can be verified that in any valid coloring, the following holds: $\forall s \in S, N[s]$ induces in $G^*[U \cup S]$ a $K_3 = \{u, v, s\}$ where $u, v \in U$. Since vertices of S and Type 3 vertices are simplicial in $G^*[U \cup S]$, any central vertex of induced P_3 in $G^*[U \cup S]$ must be an

uncolored Type 4 vertex. Particularly, the vertices of a cycle $C_{k\geq 4}$ are central vertices of induced P_3's. Moreover, an edge of induced P_3 must be heavy and the other one must be light. It's easy to see that vertices of a light edge must have different colors. The following lemma is helpful to extend coloring Γ'.

Lemma 4.14. *Let Γ'' any total valid coloring extensible from Γ' and $P = (v_1, \ldots, v_t)$ be an induced path of $G^*[U \cup S]$ such that v_1, v_t are type 4 vertices, v_1v_2 is a light edge and v_1 is a black vertex, then (i) v_iv_{i+1} is a light (heavy) edge if i is odd (even); (ii) v_i is black (white) if i is odd (even).*

Proof. We know that P is an induced path, $v_2, \ldots, v_{t-1}$ are central vertices of induced P_3's, they are also Type 4 vertices and the edges of P are light and heavy alternately. Then (i) holds because the first edge is white. On the other hand, vertices of light edges must have different colors, while the same occurs for heavy edges if one vertex is white. Since v_1 is black and v_1v_2 is a light edge, then v_2 is white and v_3 is black. Again, we can check that v_3 satisfies the same properties as v_1 and v_4 will satisfy the same properties as v_2. Therefore there is a unique valid coloring for vertices of P which consists of alternating the colors of the vertices, where (ii) v_i is black if and only if i is odd. □

We proceed by finding a minimum weight DIM on each connected component G_i of $G^*[U \cup S]$:

G_i is a chordal graph

In this case, for each single black vertex s in G_i, its neighbors u and v form an edge and we set $\omega(uv) = \infty$. In this way any non infinity weight DIM of G_i will not contain this edge and s will be a black vertex as in C'. Apply the algorithm described in the previous section that computes $DIM_\Omega(G_i)$, if existing.

G_i has an induced cycle $C_k, k \geq 4$

As it was mentioned before, C_k is formed by light and heavy edges, where each light edge is adjacent to heavy edges and vice versa.

Lemma 4.15. *[28]: Let G^* be the resulting claw–free graph and Γ the partial valid coloring obtained after Phase 3. If the subgraph of G^* induced by uncolored vertices contains an induced cycle $C_{k\geq 4}$, then k is even. Moreover, if G^* admits a black-white partition, then the vertices of C_k are colored alternately black and white along the cycle, and furthermore, by switching the colors of vertices of C_k we again obtain a valid black-white partition of G^*.*

Lemma 4.16. *G_i admits exactly two DIMs or none.*

Proof. We extend the initial coloring choosing any alternate coloring for C_k and applying propagation rules. Let Γ_i be this result coloring. We will prove that Γ_i is invalid or is a total valid coloring. Clearly, if Γ_i is invalid then G_i has no DIMs by lemma 4.15. If Γ_i is a total valid coloring, then switching the colors of vertices of C_k, we obtained another total valid coloring of G_i and they are the unique total valid colorings. Suppose that Γ_i is valid but there is some uncolored vertex u in G_i. Let $P = (v_0, \ldots, v_t = u)$ be the shortest path from a vertex v_0 in C_k. Without loss of generality we can assume that $v_0, \ldots, v_{t-1}$ are colored vertices. Clearly, P is an induced path. On the other hand, v_0v_1 is a heavy edge because v_1 must be adjacent to two consecutive vertices of C_k by the claw-freeness. Hence, v_1 must be a black vertex and $t \geq 2$. Then v_1, v_{t-1} are central vertices of induced P_3's which implies that v_1, v_{t-1} are type 4 vertices and v_1v_2 is a light edge. Clearly, $v_t = u$ must be type 3 vertex because otherwise it must be colored applying lemma 4.14. Hence, $v_{t-1}v_t$ is a heavy edge and $v_{t-2}v_{t-1}$ is a light edge which means that t is odd and v_{t-1} is white vertex. Therefore, v_t must be a black vertex which is a contradiction. Consequently, Γ_i is a total valid coloring. □

Using these two lemmas, we can determine in linear time all DIMs of G_i and return one of minimum weight (if existing).

As for the complexity of the last phase of the algorithm, observe that a cycle of length ≥ 4 of a non chordal graph can be obtained in linear time in the size of G, that is, $O(n)$ time. All the remaining steps can be completed in $O(n)$ time. It should be noted that the corresponding phase of the algorithm [28] requires $O(n^2)$ time. The main difference is that in the the presently proposed algorithm, it is sufficient to find just one induced cycle of length ≥ 4, and propagate the coloring to its connected component, whereas the algorithm [28] requires the computation of $O(n)$ such cycles, in subgraphs not necessarily disjoint of the graph. Since each of them needs $O(n)$ time, the overall complexity of the latter algorithm is $O(n^2)$.

Our proposed formulation computes $DIM_\Omega(G)$ in $O(n)$ time. Observe that through the algorithm the input graph is modified, however the changes do not alter the value of the $DIM_\Omega(G)$ solution. As for the actual minimizing DIM, itself, there is no difficulty retrieving it in $O(n)$ time, by backwards computation. Note that the algorithm can be easily adapted to be robust in the sense of [92], that it does not require the input graph G to be claw-free.

4.5 Counting DIMs

We design an algorithm to count DIMs which uses the first three phases of the previous algorithm. We proceed to explain the new *Phase 4* in order to obtain the number of DIMs of G. For each connected component G_i, we count the number of DIMs in G_i. Thus the number of DIMs in G is the product of these amounts. The case where G_i is not chordal is easy since number of DIMs is 0 or 2 (see lemma 4.16). For the case where G_i is chordal we give a more detailed algorithm. From [28], using the proof of Lemma 12, we can state that: **(i)** Each vertex of G_i belongs to a triangle and any two triangles of G_i are disjoint; **(ii)** Each triangle has at most one colored vertex, each colored vertex of G_i has type 4 in G and becomes of type 3 in G_i ; **(iii)**

By contracting each triangle into a single vertex we obtain a tree. A triangle that becomes a leaf in this tree is called ***leaf triangle***.

Each **leaf triangle** has two type 3 vertices (one of which may be *single*) and one type 4 vertex. An edge with two vertices of type 3 is a ***bottom edge***, otherwise is a ***non-bottom edge***.

Lemma 4.17. *A total valid coloring, extending current coloring, has at most one non-bottom edge.*

Proof. Suppose that there exists a DIM M that contains vw and $v'w'$, two different *non-bottom edges*, with $T = \{v, w, z\}$ and $T' = \{v', w', z'\}$ the triangles where those edges belongs. Suppose w.l.o.g. v, v' are vertices of type 4. The shortest path $P = (v = v_1, v_2, \ldots, v_k = v')$ from v to v', is an induced path. Clearly, v_1v_2 and $v_{k-1}v_k$ are light edges. By Lemma 4.14, v or v' must be white vertex which means vw or $v'w'$ does not belong to M which is a contradiction. □

Therefore we can count DIMs of two types:

(a) DIMs that contain exactly one *non-bottom edge*: Choose one *non bottom edge* vw from a leaf triangle $T = \{v, w, z\}$ where v is of type 4 and w is not a single vertex. Color v, w with black and apply propagation rules. It can be checked that the result is a total valid coloring where vertices can be colored in a unique way. Thus for each possible leaf triangle where the *non-bottom edge* can be chosen to color with black, we have one more DIM to count.

(b) DIMs that do not contain *non-bottom edge*: Each *bottom edge* from every *leaf triangle* must be chosen (color both vertices with black) and then extend the coloring using propagation rules. If the result is not a valid coloring then there is no DIM of this type, otherwise we get a valid coloring: if it is total then return one DIM else we can recursively count the DIMs from this smaller instance. Clearly, each time the algorithm calls recursively will multiply the results obtained. It is easy to note that this whole process takes linear time.

Since the original graph G has not infinite weight DIMs, they must be avoided when counting. Before Phase 4, we can check if there is an already chosen edge with infinite weight, in this case, there is NO DIM. The infinite weight edges assigned during *Phase 1* can be only *bottom edges*.

Let B the set of *bottom edges* with infinite weight. Clearly, any DIM M must choose some edge of each leaf triangle, only one of these edges can be non-bottom edge which means $|M \cap B| \geq |B| - 1$: (i) If $|B| \geq 2$ then there is NO DIM since any DIM M has infinity weight. (ii) If $|B| = 1$ which means there is only one infinity bottom edge vw of a leaf triangle $T = \{u, v, w\}$. Clearly all DIMs of type (b) contain vw and they have infinity weight. There are at most two DIMs of type (a) which do not contain vw. They must contain some non-bottom edge (uv or uw) of T. There are exactly two non infinity weight DIMs if v, w are uncolored vertices. Otherwise, v or w is a single black vertex which implies that there is exactly one non infinity weight DIM. (iii) If $B = \emptyset$, then NO DIM has infinite weight, thus all DIMs can be counted recursively as we explained before.

Chapter 5

Efficient and Perfect Domination

5.1 Preliminaries

The efficient edge domination problem (also known as Dominating Induced Matching, and denoted as DIM) was studied in the previous chapter. The notion of this problem comes from more general problems, which are the efficient and perfect domination. Given a graph $G = (V, E)$, a perfect dominating set is a subset of vertices $V' \subseteq V(G)$ such that each vertex $v \in V(G) \setminus V'$ is dominated by exactly one vertex $v' \in V'$. An **efficient dominating set** is a perfect vertex dominating set V' where V' is also an independent set. Every graph G contains a perfect dominating set, for instance, take $V(G)$. But not every graph contains an efficient vertex dominating set. These problems consists in searching the sets with minimum number of vertices. All of them are NP-hard, even when restricted to some particular graph families. The weighted version of these problems, where each vertex v has a weight assigned $\omega(v)$, consists on finding a perfect vertex dominating set where the sum of the weights is minimum. We denote these problems as Minimum Weighted Perfect Vertex Domination (MWPVD), Minimum Weighted Efficient Vertex Domination (MWEVD). We denote the edge-versions of these problems as Minimum Weight Perfect Edge Domination (MWPED) and Minimum Weight Efficient Edge Domination (MWEED). Note that for these

edge-versions the dominating set consists of edges instead of vertices, hence the weights are on the edges, and the adjacency of two edges is defined as two edges that share a vertex.

In this chapter we show results for the weighted perfect domination problem, and for the efficient domination problem, restricted to circular-arc graphs.

5.2 Circular-Arc graphs

We give some auxiliary definitions and properties for circular-arc graphs.

The following definitions and results come from [47, 48].

A circular-arc (CA) model for G is a pair (C, A), where C is a circle and $\mathcal{A}$ is a collection of arcs of C, such that each arc $A_i \in \mathcal{A}$ corresponds to a vertex $v_i \in V(G)$, and A_i, Aj intersect precisely when v_i, vj are adjacent, $i \neq j$. A circular-arc (CA) graph is one admitting a CA model. When traversing the circle C, we will always choose the clockwise direction. If s, t are points of C, write (s, t) to mean the arc of C defined by traversing the circle from s to t. Call s, t the extremes of (s, t), while s is the start and t the end of the arc. For $A_i \in \mathcal{A}$, write $A_i = (s_i, t_i)$. Without loss of generality, all arcs of C are considered as open arcs, no two extremes of distinct arcs of A coincide and no single arc entirely covers C.

A Helly circular-arc (HCA) graph G is a CA graph admitting a CA model whose arcs satisfy the Helly property. That is, every pairwise intersecting subfamily of arcs of A contains a common point. Such a model is called a Helly circular-arc (HCA) model for G.

Given a circular-arc model $\mathcal{M} = (C, \mathcal{A})$ where $\mathcal{A} = \{A_1 = (s_1, t_1), \ldots, A_n = (s_n, t_n)\}$, if an arc $A \in \mathcal{A}$ contains some point $p \in C$ then we say that A is an arc of p. Denote by $\mathcal{A}(p)$ the collection of arcs of p. We say two points $p, p' \in C$ are equivalent if $\mathcal{A}(p) = \mathcal{A}(p')$. The $2n$ extreme points from the n arcs of $\mathcal{A}$ divide the circle C in $2n$ segments of the following types: (i) (s_i, t_j) (ii) $[t_i, t_j)$ (iii) $(s_i, s_j]$ (iv)

$[t_i, s_j]$. We say the segments of type (i) are *intersection segments*. It is easy to see that all points inside one of the $2n$ segments are equivalent.

Corollary 5.1. *[47, 48] There are at most $2n$ distinct $\mathcal{A}(p)$.*

Given a circular-arc model $\mathcal{M} = (C, \mathcal{A})$, the following algorithms can be achieved in $O(n)$ time:

- Find a universal arc. A universal arc has common intersection with every arc from $\mathcal{A}$.
- Find two arcs A_i, A_j such that $A_i \cup A_j = C$.
- Find three arcs A_i, A_j, A_k such that $A_i \cup A_j \cup A_k = C$.
- Find a point $p \in C$ such that $|\mathcal{A}(p)|$ is maximum.
- Find a point $p \in C$ such that $|\mathcal{A}(p)|$ is minimum.
- If $max_{p \in C}|\mathcal{A}(p)| = 2$ and $min_{p \in C}|\mathcal{A}(p)| = 1$ and do not exists two arcs from $\mathcal{A}$ that cover the entire circle C then exists an induced cycle C_k with $k \geq 3$ such that arcs corresponding to the vertices cover the circle C in $\mathcal{M}$ and the rest of the arcs from $\mathcal{A}$ are pairwise disjoint and are contained in exactly one of the arcs from C_k. Thus each arc is either part of the C_k or a leaf with a parent arc from C_k. It is possible to identify each arc from the C_k.

Note that $max_{p \in C}|\mathcal{A}(p)| > min_{p \in C}|\mathcal{A}(p)|$ if $\mathcal{A} \neq \emptyset$. For instance, let $A = (s, t) \in \mathcal{A}$, $\mathcal{A}(s) \subset \mathcal{A}(s + \epsilon)$ and $A \in \mathcal{A}(s + \epsilon) \setminus \mathcal{A}(s)$. Hence $max_{p \in C}|\mathcal{A}(p)| \geq |\mathcal{A}(s + \epsilon)| > |\mathcal{A}(s)| \geq min_{p \in C}|\mathcal{A}(p)|$.

Lemma 5.1. *[47, 48] Given a CA model $\mathcal{M} = (C, \mathcal{A})$, if there are no two or three arcs of $\mathcal{A}$ that cover the entire circle C then $\mathcal{M}$ is an HCA model.*

In the following four sections we consider each of the four variants of the mentioned problems for circular-arc graphs, we show the current state of the art and we propose

linear time algorithms to solve them, except for the MWEVD where there already exists a linear time algorithm. We assume that our input is a circular-arc graph G and a weight function ω over the vertices or edges depending on the problem we are solving. For simplicity, we may already use the circular-arc model for G. For these cases, there is an implicit previous step which is applying a linear time algorithm [82] in order to obtain a circular-arc model for G.

For MWEVD and MWEED, we consider the function ω non-negative. Since every efficient vertex (edge) dominating set of a graph contains exactly the same number of vertices (edges), it is possible to subtract to values from ω the minimum weight of the vertices (edges). This does not work for the MWPVD and MWPED cases.

5.3 Minimum weighted efficient vertex domination

The minimum weighted efficient vertex domination problem (MWEVD) on a graph G can be expressed as an instance of the minimum weighted dominating set problem (MWDS) on the same graph G making some minor adjustments on the way described in [22] for unweighted version of MWEVD. There is an $O(n+m)$ time algorithm to solve MWDS for circular-arc graphs [30]. Hence, MWEVD can be solved for circular-arc graphs in linear time.

5.4 Minimum weighted efficient edge domination

The Minimum weighted efficient edge domination problem (MWEED) for a graph G is equivalent to either:

(1) MWEVD for the line graph $L(G)$

(2) MWIS for the square of the line graph $L^2(G)$

If G is a circular-arc graph, then the graph $G' = L^2(G)$ is also a circular-arc graph [60]. The graph G' has exactly m vertices and up to $O(m^2)$ edges. The algorithm from [92] solves MWIS problem in linear time for circular-arc graphs. We show an upper bound on the number of edges for circular-arc graphs that admit a DIM. It is known that a graph that admits a DIM is K_4-free. This property imposes a bound on the number of edges, thus the algorithm complexity bound can be improved from $O(m^2)$ to $O(n^2)$.

We know from Lemma 4.11, every K_4-free graph G where $n \geq 2$ such that G is either chordal or interval graph has at most $2n - 3$ edges.

Lemma 5.2. *Every K_4-free graph G where $|V(G)| \geq 2$ such that G is circular-arc graph has at most $2n$ edges.*

Proof. If G is an interval graph the property is satisfied by lemma 4.11. Thus, suppose $|V(G)| \geq 4$ and G is not an interval graph. If exist p such that $|\mathcal{A}(p)| = 0$ then G is an interval graph and if $|\mathcal{A}(p)| \geq 4$ then G contains a K_4. In consequence, $1 \leq |\mathcal{A}(p)| \leq 3$ for any point p. Let p be the point with maximum value of $|\mathcal{A}(p)|$. Clearly, p belongs to an intersection segment (s_i, t_j) where $|\mathcal{A}(s_i)| = |\mathcal{A}(p)| - 1$. Hence, $|\mathcal{A}(p)| \geq 2$ and we can restrict the study to $2 \leq |\mathcal{A}(p)| \leq 3$. It is easy to see that at least one of these exists. We analyze each possible case:

- $|\mathcal{A}(p)| = 2$: Let (s_i, t_j) the intersection segment that contains p:

    1. $i = j$: In this case $\mathcal{A}(p) = \{A_i, A_k\}$ where A_k contains arc A_i. If we cut arc A_k at point s_i converting it in two different arcs $(s_k, s_i - \epsilon)$ and $(s_i + \epsilon, t_k)$ we get an interval graph G' with $n + 1$ intervals. By Lemma 4.11 the number of edges of G' is at most 2(n+1)-3 = 2n-1 edges. Given that G' has at least the same number of edges that G we conclude that $|E(G)| \leq 2n - 1$.
    2. $i \neq j$: In this case $\mathcal{A}(p) = \{A_i, A_j\}$. If we replace A_i by $(s_i, p - \epsilon)$ and A_j by $(p + \epsilon, t_j)$, we get an interval graph G' with n vertices. However

the edge that connects A_i, A_j has been lost unless $A_i \cup A_j$ cover the entire circle. Therefore $|E(G')| \leq 2n-3$ and $|E(G)| \leq 2n-3+1 = 2n-2$.

- $|\mathcal{A}(p)| = 3$:

  1. $i = j$: In this case $\mathcal{A}(p) = \{A_i, A_k, A_p\}$ where A_k, A_p contain arc A_i. If we cut arcs A_k, A_p at point s_i, converting each one into two different arcs, we get an interval graph G' with $n+2$ intervals and at least one more edge since A_k, A_p have been converted to two different edges. By Lemma 4.11 the number of edges of G' is at most $2(n+2)-3 = 2n+1$, hence $|E(G)| \leq 2n$
  2. $i \neq j$: In this case $\mathcal{A}(p) = \{A_i, A_j, A_k\}$ where A_k contains the segment (s_i, t_j). We can replace A_i by $(s_i, p-\epsilon)$ and A_j by $(p+\epsilon, t_j)$ (and may lose an edge unless $A_i \cup A_j$ cover the entire circle). We cut the arc A_k in either s_i or t_j, any one works. We get an interval graph G' with $n+1$ vertices and at least $|E(G)|-1$ edges. By lemma 4.11 the number of edges of G' is at most $2(n+1)-3 = 2n-1$. Hence $|E(G)| \leq 2n$.

The graph $\overline{3K_2}$ has 6 vertices and 12 edges, hence it shows tightness of the bound. Therefore any algorithm to solve DIM problem can first check the amount of edges to ensure the existence of a DIM. Since $m \in O(n)$ then any $O(n+m)$ time algorithm for circular-arc graphs can be easily converted to an $O(n)$ time algorithm. □

In the previous chapter, a linear-time algorithm to solve MWEED for general graphs given a fixed dominating set was presented.

If there exists a set of at most three arcs that cover the entire circle, then there is a dominating set of size at most 3, thus the problem can be solved using the mentioned algorithm in linear time. Hence we can assume a model M without a set S that cover

the entire circle such that $|S| \leq 3$. Therefore M is a Helly circular-arc (HCA) model and the original graph G is HCA.

We analyze the model M:

(a) $max_{p \in C}|\mathcal{A}(p)| \geq 4$**:** Then G is not K_4-free, hence it does not admit a DIM

(b) $min_{p \in C}|\mathcal{A}(p)| = 0$**:** Then M is an interval model. Algorithm from [80] can be applied

(c) $max_{p \in C}|\mathcal{A}(p)| = 2$**:** We refer this case to subsection 5.4.1

(d) $max_{p \in C}|\mathcal{A}(p)| = 3$**:** We refer this case to subsection 5.4.2

Each of these cases can be implemented in linear time.

5.4.1 $max_{p \in C}|\mathcal{A}(p)| = 2$

Since $max_{p \in C}|\mathcal{A}(p)| > min_{p \in C}|\mathcal{A}(p)|$, $min_{p \in C}|\mathcal{A}(p)| = 1$. We can locate an induced cycle $C_{k \geq 4}$ on G such that arcs from those vertices cover the entire circle C in the model M, and the rest of arcs from A are pairwise-disjoint and are contained in exactly one arc from C_k. If a vertex $v \in C_k$ is connected with a set of pendant vertices W, then v should be colored with black in order to get a coloring of the graph that represents a valid DIM. Note that the only edge that belongs to a minimum weighted DIM will be an edge of minimum weight between v and W. Therefore, all edges between v and $w \in W$ such that its weight is not minimum can be removed from the graph G in order to obtain G'. Every minimum weighted DIM from G is a valid minimum weighted DIM in G'. Note that if exist k edges with minimum weight, then any $k-1$ of those edges can be erased. In the new graph G' every vertex $v \in C_k$ contains at most one pendant vertex w in his neighborhood. We know that HCA is an hereditary property, G' is still an HCA graph, and every clique from G' is a K_2, hence $K(G') = L(G')$. The graph $K(G')$ is HCA if G' is HCA and it contains $O(n)$ vertices and $O(n)$ edges since maximum $\Delta(L(G')) \leq 4$. The $K(G')$ model can be

obtained within $O(n)$ time [76]. Therefore MWEED can be solved for G', by solving MWEVD for $L(G')$, which can be done in linear time.

5.4.2 $max_{p \in C}|\mathcal{A}(p)| = 3$

Denote $\mathcal{A}(p) = \{A_1, A_2, A_3\}$. Then v_1, v_2, v_3 form a triangle in G, which means that any DIM of G must contain exactly one of the three edges of this triangle: $\{v_1v_2, v_2v_3, v_3v_1\}$. By removing point p from C (every arc (s,t) containing p is now two disjoint arcs $(s, p-\epsilon)$ and $(p+\epsilon, t)$), we obtain an interval graph G_p, in which the vertices v_1, v_2, v_3 of G correspond to two triangles formed by u_1, u_2, u_3 and $u_{n+1}, u_{n+2}, u_{n+3}$, respectively. So, any DIM of G_p must contain one edge of each of these triangles. Furthermore, our task would become simpler if u_1, u_2, u_3 and $u_{n+1}, u_{n+2}, u_{n+3}$ would correspond to triangles having neither common nor adjacent vertices. The latter condition is fulfilled because there are no two arcs that cover the entire circle.

Lemma 5.3. *The triangles* $\{u_1, u_2, u_3\}$ *and* $\{u_{n+1}, u_{n+2}, u_{n+3}\}$ *have neither common nor adjacent vertices.*

Proof. Clearly, $u_1, u_2, u_3, u_{n+1}, u_{n+2}, u_{n+3}$ are six different vertices. Hence there are no common vertex in both triangles. Suppose that u_i is adjacent to u_{n+j}, $1 \leq i, j \leq 3$. Consider the following cases.

(a) $i = j$**,** which means A_i covers the circle C, that is absurd.

(b) $i \neq j$**,** in this case, A_i and A_j cover the circle which is a contradiction.

□

So, we now consider that the triangles u_1, u_2, u_3 and $u_{n+1}, u_{n+2}, u_{n+3}$ contain neither common nor adjacent vertices. We will apply algorithm from [80] to G_p, which solves MWEED for chordal graphs (hence interval graphs). The algorithm will be executed three times. In each of the applications, the graph G_p remains the same,

but the weighting changes, to Ω_1, Ω_2 and Ω_3, respectively. In order to avoid edges of triangles $\{u_1, u_2, u_3\}$ and $\{u_{n+1}, u_{n+2}, u_{n+3}\}$ incident to u_i or u_{n+i} to be included in the DIM, in the weighting Ω_i, $1 \leq i \leq 3$, we assign to each of these edges a high weight, for instance more than twice the sum of all weights of the edges of G. All the other edges of G_p remain the same as in the corresponding edges of G. If the value of the minimum weighted DIM of G_p is above that high weight assigned to the edges we want to avoid, then we know that G contains no DIM. Otherwise, we have solved our problem, and we only need to subtract the duplicated weight among the edges of the triangles u_1, u_2, u_3 and $u_{n+1}, u_{n+2}, u_{n+3}$ which is part of the solution for G_p, but not for G.

The following are the formal definitions of the weights. Let A_1, A_2, A_3 be the arcs containing p. Assign to G_p the weighting Ω_i, $1 \leq i \leq 3$, which defines a weight $\omega_i(u_j u_k)$ for each edge $u_j u_k \in E(G_p)$, as follows.

For $1 \leq i, j \leq 3 \leq k \leq n$ and $u_j u_k \in E(G_p)$,
$\omega_i(u_j u_k) := \omega(v_j v_k)$

For $1 \leq i, j \leq 3 \leq k \leq n$ and $u_{n+j} u_k \in E(G_p)$,
$\omega_i(u_{n+j} u_k) := \omega(v_j v_k)$

For $1 \leq i, j, k \leq 3$, $i \neq j$, $j \neq k$ and $k \neq i$,
$\omega_i(u_j u_k) := \omega_i(u_{n+j} u_{n+k}) := \omega(v_j v_k)$

For $1 \leq i, j \leq 3$ and $i \neq j$,
$\omega_i(u_i u_j) := \omega_i(u_{n+i} u_{n+j}) := 1 + 2 \cdot \sum_{(u,v) \in E(G_p)} \omega(uv)$

Lemma 5.4. *Let* $max|\mathcal{A}(p)| = 3$, $\mathcal{A}(p) = \{A_1, A_2, A_3\}$. *Then*

$$dim_\Omega(G) = \begin{cases} \infty & min_{1\le i\le 3}\{dim_{\Omega_i}(G_p)\} > 2\cdot \sum\limits_{(u,v)\in E(G_p)} \omega(uv)\} \\ min_{1\le i\le 3}\{dim_{\Omega_i}(G_p) - \omega_i(u_ju_k)\}^\dagger & \text{otherwise} \end{cases}$$

$\dagger$ where $1 \le j, k \le 3; i \ne j \ne k \ne i$

Proof. Let $\mathcal{A}(p) = \{A_1, A_2, A_3\}$. Cut the circle at point p and consider the model $\mathcal{M}$ of the interval graph G_p. For lemma 5.3, $\{u_1, u_2, u_3\}$ and $\{u_{n+1}, u_{n+2}, u_{n+3}\}$ are triangles with neither common nor adjacent vertices.

Suppose G has a DIM M, weighted by Ω, having total weight $\omega' \le 2\sum\limits_{(u,v)\in E(G_p)} \omega(uv)$. We know that exactly one edge of the triangle $\{v_1, v_2, v_3\}$, say v_1v_2, belong to M. Then choose the weighting Ω_3 for G_p and consider the following subset of edges $M_p \subseteq E(G_p)$.

$$M_p = \{u_1u_2, u_{n+1}u_{n+2}\} \cup \{u_iu_j \in E(G_p) | v_iv_j \in M, 3 < i, j \le n\}$$

We claim the M_p is a DIM for G_p. Since M is a matching, M_p is clearly so. Assume by contrary that it is not induced, and let the violating edge be u_iu_j, where u_i, u_j are vertices incident to distinct edges of M_p. Since M is an induced matching of G, it follows that $i \in \{1, 2, 3\}$ and $j \in \{n+1, n+2, n+3\}$. The latter means that u_1, u_2, u_3 and $u_{n+1}, u_{n+2}, u_{n+3}$ have a common or an adjacent vertex, a contradiction. Consequently, M_p is indeed an induced matching. It remains to show that it is dominating. Let S be the subset of vertices of G_p, not incident to the edges of M_p. Let $u_i, u_j \in S$, $i \ne j$. Because M is a dominating matching, the only possibility for u_i, u_j to be adjacent is $i = 3$ and $j = n + 3$, which contradicts A_3 not covering the circle. Then M_p is a DIM having weight equal to $\omega' + \omega(v_1, v_2)$.

Conversely, assume that G_p has a DIM weighted by, say Ω_3, having weight $\le 2\sum\omega$. We know that exactly one edge of each of the triangles $\{u_1, u_2, u_3\}$ and

$\{u_{n+1}, u_{n+2}, u_{n+3}\}$ belong to M_p. Furthermore, the edges of these triangles which are incident either to u_3 or u_{n+3} all have weight $> 2\sum\omega$. Therefore any dominating set of edges with weight $\leq 2\sum\omega$ contains the edges u_1u_2 and $u_{n+1}u_{n+2}$. Let

$$M = \{v_1v_2\} \cup \{v_iv_j \in E(G) | u_iu_j \in M_p, 3 < i, j \leq n\}$$

We claim that M is a DIM of G. Clearly, for any $v_i, v_j \in V(G)$, $1 \leq i \leq 3$ and $3 < j \leq n$, $v_iv_j \in E(G)$ if and only if $u_iu_j \in E(G_p)$ or $u_{n+i}u_j \in E(G_p)$. Consequently, M_p being a DIM of G_p implies that M is a DIM of G. Furthermore, the weight of M is precisely the weight of M_p less $\omega_3(u_1u_2)$, since it was counted twice in M_p. The lemma follows. □

5.5 Minimum weighted perfect vertex Domination

Given a weighted graph G with weight function $\omega : V(G) \to \mathcal{R}^+$, the goal is to find the perfect vertex dominating set of minimum weight. If G contains a universal vertex then the following lemma can be used in order to obtain the solution. In [31], an $O(n+m)$ time algorithm to solve MWPVD for interval graphs was presented. The same paper shows the only known algorithm to solve MWPVD for circular-arc graphs in $O(n^2 + nm)$ time. Note that any efficient vertex dominating set is also a perfect vertex dominating set. The following lemma along with the procedure derived from it can be applied to graphs with $\omega : V(G) \to \mathcal{R}$ (i.e. negative weights included).

Lemma 5.5. *Given a graph $G = (V, E)$ where $u_1 \in V(G)$ is a universal vertex:*

1. *If G contains another universal vertex $u_2 \neq u_1$, then the unique perfect vertex dominating sets of G are: V, $\{u_i\}$ where u_i may be any universal vertex.*

2. *If u_1 is the unique universal vertex from G, then every perfect vertex dominating set of G contains u_1. Moreover, each solution can be computed on the following*

way: Let $\{G_1 = (V_1, E_1), \ldots, G_k = (V_k, E_k)\}$ *the connected components from* $G \setminus \{u_1\}$. *Then*

$D \subseteq V$ *is a perfect dominating set* $\iff u_1 \in D \wedge (D \cap V_i = \emptyset \vee D \cap V_i = V_i)$

Proof. The proof is separated in two cases:

1. If G contains universal vertex $u_2 \neq u_1$:

   It is clear that V and each universal vertex $\{u_i\}$ are perfect vertex dominating sets from G. Assume there exists another perfect vertex dominating set D. If $|D| = 1$ then it should be a universal vertex and was one of the above mentioned sets. Hence D contains at least two vertices and there is at one least vertex $w \notin D$. Since $|D| > 1$ then every universal vertex u_i should belong to D, otherwise the vertex u_i will be dominated by more than one vertex inside D, which contradicts the definition of perfect vertex domination. Moreover, every vertex $w \notin D$ will be dominated by at least two universal vertices from D, then again, this contradicts the definition of the set D. Therefore, there is no other perfect vertex dominating set.

2. If u_1 is the unique universal vertex from G:

   $\implies$ Assume there exists a perfect vertex dominating set D that does not contains u_1. It is clear that D should have at least two vertices since there is not another universal vertex, but then the vertex u_1 is dominated by $|D| > 1$ vertices, which is absurd. Thus every perfect vertex dominating set contains u_1. Suppose D is a perfect vertex dominating set such that $v \in D$, $w \notin D$ and $v, w \in V_i$. Given that v, w belongs to the same connected component G_i, there exists a pair of adjacent vertices $v', w' \in V_i$ such that $v' \in D$ and $w' \notin D$. Thus w' is dominated by u_1 and v', which is a contradiction. Thus every perfect vertex dominating set D contains u_1 and satisfies $D \cap V_i = V_i \ \vee \ D \cap V_i = \emptyset$ for each connected component V_i.

   $\impliedby$ Let D be a subset of vertices such that for every connected component V_i, either $D \cap V_i = \emptyset$ or $D \cap V_i = V_i$ and $u_1 \in D$. It is easy to see that D dominates

$V(G)$ since $u_1 \in D$. Let w be a vertex such that $|N(w) \cap D| > 1$. Assume w.l.o.g. $w \in V_i$. If $D \cap V_i = \emptyset$ then $N(w) \cap D = \{u_1\}$. Otherwise $D \cap V_i = V_i$, hence $w \in D$. Therefore, every vertex $w \in V(G)$ is either in D, or dominated by u_1. We conclude D is a perfect dominating set.

□

If G contains at least two universal vertices, then the amount of perfect vertex dominating sets is at most $O(n)$, and the one with minimum weight can be obtained in linear time. In case G contains exactly one universal vertex the minimum weighted perfect vertex dominating set can be obtained with the following procedure:

$D := \{u_1\}$. We define the weight of a set of vertices as the sum of the weights of its vertices. If the weight of a connected component G_i of $G \setminus \{u_i\}$ is negative, then $D := D \cup V_i$.

5.5.1 MWPVD algorithm for CA-graphs

Given a circular-arc graph G, a circular-arc model $\mathcal{M}$ from G can be obtained in $O(n + m)$ time and universal arcs can be identified in $O(n)$ time. If a universal arc exists, we can solve the problem using the procedure mentioned above. Thus we assume the graph G does not contain a universal vertex.

It is possible to solve MWEVD in linear time for circular-arc graphs. Hence we can save the best efficient vertex dominating set as a candidate solution, and search for the perfect vertex dominating sets that are not efficient vertex dominating sets.

We determine in $O(n)$ time the point p such that $|\mathcal{A}(p)|$ is minimum:

(i) $|\mathcal{A}(p)| = 0$**,** In this case $\mathcal{M}$ is an interval model and G an interval graph. A linear-time algorithm [31] can be applied to solve MWPVD.

(ii) $|\mathcal{A}(p)| \geq 2$**,** thus for every point $p' \in C$, $|\mathcal{A}(p')| > 2$. Let D be a perfect vertex dominating set which is not an efficient vertex dominating set, and with

minimum weight. There should be two adjacent vertices $v, w \in D$, otherwise D is an efficient vertex dominating set. Let A_v, A_w the corresponding arcs. Note that $A_v \cap A_w \neq \emptyset$, and let $q \in A_v \cap A_w$. For any arc $A_z \in \mathcal{A}(q)$ the vertex z (corresponding to A_z) should belong to D, otherwise (otherwise it will be dominated by two vertices). Let $q' \in C$ the first point from q in clock-wise order such that $\mathcal{A}(q') \neq \mathcal{A}(q)$ (hence q and q' belong to different segments). We will show that for any arc $A_z \in \mathcal{A}(q')$, the vertex z corresponding to A_z should be in D We consider two cases according to segment finalization q.

- If it has open-end at t_u, then $\mathcal{A}(q') = \mathcal{A}(q) \setminus \{A_u\}$. Thus, every arc $A_z \in \mathcal{A}(q') \subset \mathcal{A}(q)$, the vertex z corresponding to $A_z \in D$.
- If it has close-end at s_u, then $\mathcal{A}(q) = \mathcal{A}(q') \setminus \{A_u\}$. Thus $\mathcal{A}(q')$ contains arcs from $\mathcal{A}(q)$ and an additional arc A_u. If A_u is adjacent to arcs from $\mathcal{A}(q)$ and $|\mathcal{A}(q)| > 1$ then A_u should be in D, since $\mathcal{A}(q)$ belongs to D and A_u could not have more than one adjacent from D, unless is part of D.

Let $q := q'$ and apply iteratively the same procedure $2n$ times, this is the amount of different segments. This shows that all vertices are in D. Thus $D = V$, and the solution will be the best among $\sum_{v \in V} \omega(v)$ and the best solution previously found.

(iii) $|\mathcal{A}(p)| = 1$**,** In this case $\mathcal{A}(p) = \{A_v\}$ where $A_v = (s_v, t_v)$ corresponds to a vertex $v \in V$. Recall that $\mathcal{A}$ has no universal arc. We generate three interval models ($\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$) from $\mathcal{M}$ and replace A_v with $A_{v^-} = (s_v, p - \epsilon)$ and $A_{v^+} = (p + \epsilon, t_v)$, adding arcs for each one of them and modifying the weight function as we describe next:

($\mathcal{M}_1$) We replace A_v with A_{v^-} and A_{v^+}. We also add $A_{w^-} = (p - 1.5 * \epsilon, p - 0.5 * \epsilon)$ and $A_{w^+} = (p + 0.5 * \epsilon, p + 1.5 * \epsilon)$, assigning weights to

the vertices that corresponds to new arcs: $\omega(v^-) := \omega(v^+) := 0.5 * \omega(v)$, $\omega(w^-) := \omega(w^+) := \infty$.

($\mathcal{M}_2$) We replace A_v with A_{v^-} and A_{v^+}. We also add $A_{w^-} = (p - 1.5 * \epsilon, p - 0.5 * \epsilon)$, assigning weights to the vertices that corresponds to the new arcs: $\omega(v^-) := \omega(v^+) := \infty$, $\omega(w^-) := 0$.

($\mathcal{M}_3$) We replace A_v with A_{v^-} and A_{v^+}. We also add $A_{w^+} = (p + 0.5 * \epsilon, p + 1.5 * \epsilon)$, assigning weights to the vertices that corresponds to the new arcs: $\omega(v^-) := \omega(v^+) := \infty$, $\omega(w^+) := 0$.

It is easy to see that described models are interval models. Hence MWPVD can be solved for those three interval models using a linear time algorithm from [31].

We show how to map the perfect vertex dominating set D_i from $\mathcal{M}_i$, $1 \leq i \leq 3$, to a perfect vertex dominating set D from G maintaining the weight.

- The weight of D_1 is bounded, thus $w^-, w^+ \notin D_1$. Since both are leaves, they should be dominated by their parents v^- and v^+, respectively. Then $v^-, v^+ \in D_1$. In this case, we take $D = (D_1 \setminus \{v^-, v^+\}) \cup \{v\}$. Note that the weight of D and D_1 is the same since $\omega(v^-) + \omega(v^+) = \omega(v)$.

  Each vertex from $v \in G$ such that $v \notin D$ is dominated by exactly one vertex from D_1. Each dominating vertex remains except for v^- and v^+. But vertices dominated by v^- and v^+ are now dominated by v. Hence D is a perfect vertex dominating set of G.

- The weight of D_2 is bounded, hence $v^-, v^+ \notin D_2$. If w^- is a leaf and $v^- \notin D_2$, then $w^- \in D_2$. The vertex v^- is dominated by w^-, thus the rest of the neighbors of v^- are not in D_2.

  In this case, let $D = D_2 \setminus \{w^-\}$. Note that D and D_2 has the same weight since $\omega(w^-) = 0$.

Each vertex $w \notin D$ is dominated by exactly one vertex from D. All these vertices, except for v were dominated by vertices from D_2 which remains at D, except for w^-, but this vertex dominates only v^- which is not a vertex from G. v^+ is dominated by a vertex from $D = D_2 \setminus \{w^-\}$. Clearly, this dominating vertex dominates v and is the unique vertex that dominates v. As a consequence, D is a perfect vertex dominating set of G.

- The weight of D_3 is bounded, this case is symmetric to the previous case. Let $D = D_3 \setminus \{w^+\}$ be a perfect vertex dominating set of G with the same weight of D_3.

Now the perfect vertex dominating set D from G should be mapped to a perfect vertex dominating set with the same weight of D in some model $\mathcal{M}_i$, $1 \leq i \leq 3$.

The following describes the mapping:

- If $v \in D$, $D_1 = (D \setminus \{v\}) \cup \{v^-, v^+\}$ in the model $\mathcal{M}_1$. Then D_1 and D has the same weight. D_1 is a perfect vertex dominating set in $\mathcal{M}_1$, unless there exists a vertex $z \notin D_1$ such that is dominated by v^- and v^+. In this case, the arcs A_v and A_z (arc corresponding to z) covers the entire circle C. Clearly, $z \notin D$ and is dominated by $v \in D$. But D is a perfect vertex dominating set of G, thus v is a universal vertex. If this is not the case, then exists a vertex w which is not adjacent to v and the corresponding arc A_w should be contained in $A_z \setminus A_v$, but since z is dominated by v, all the intersecting arcs with z corresponds to vertices outside D. Then w cannot be dominated by any of the vertices from D because every intersecting arc of A_w intersects A_z. Absurd. Therefore v is a universal vertex, but it contradicts the hypothesis that no universal vertices exists at G. Hence, it does not exists the vertex z and D_1 is a perfect vertex dominating set of $\mathcal{M}_1$.

- If $v \notin D$, v is dominated by $z \in D$ and $t_v \in A_z$, where A_z corresponds to vertex z. It is clear that $A_v \not\subset A_z$ since $p \in A_v \setminus A_z$. In this case, $D_2 = D \cup \{w^-\}$.

Again, D and D_2 has the same weight. If D_2 is not a perfect vertex dominating set of $\mathcal{M}_2$, then v^- is dominated by $w^- \in D_2$ and another vertex $u \in D_2$. Moreover, $u \in D$ and dominates v in G but v was dominated by $z \in D$. Hence $z = u$ and A_z and A_v covers the entire circle C. Applying the same reasoning of the previous case we can conclude that z is a universal vertex and contradicts the hypothesis that G do not contain universal vertices. Therefore, D_2 is a perfect vertex dominating set in $\mathcal{M}_2$.

- If $v \notin D$, v is dominated by $z \in D$ and $s_v \in A_z$. It's symmetric to the previous case, we obtain that $D_3 = D \cup \{w^+\}$ is a perfect vertex dominating set in $\mathcal{M}_3$ with the same weight that D.

5.6 Minimum weighted perfect edge domination

We give an $O(n + m)$ time algorithm to solve MWPED for circular-arc graphs. To the best of our knowledge there is no known polynomial time algorithm to solve this problem on circular-arc graphs, while it is mentioned in [80] MWPED, where NP-completeness of unweighted version of this problem for bipartite graphs is proved.

They also showed polynomial time algorithms for some restricted graph families:

Theorem 5.1. *[80] There is an $O(n+m)$ time algorithm to solve MWPED on chordal graphs*

Corollary 5.2. *[80] There is an $O(n + m)$ time algorithm to solve MWPED on interval graphs*

Definition 5.1. *Given a graph $G = (V, E)$ and a perfect edge dominating set $E' \subseteq E$ from G, we denote $D = \{v \in V : vw \in E'\}$ the vertices adjacent to an edge from E'. We can define the following 3-coloring for the vertices of G: The black vertices $B = \{v \in D : N[v] \subseteq D\}$, the gray vertices $R = D \setminus B$ and the white vertices $W = V(G) \setminus D$.*

The following properties can be easily checked:

(P1) Each grey vertex has exactly one non-white neighbor while the rest of his neighborhood are white vertices. (the gray vertex has degree at least 2)

(P2) If $v \in W$, then $N(v) \subseteq R$. Hence W is an independent set.

It is easy to see that for any 3-coloring of vertices that satisfies properties (P1) and (P2) $E' = \{vw \in E : vw \in B \cup R\}$ is a perfect edge dominating set and for any 3-coloring of G that satisfies (P1) and (P2), if it contains K_p, with $p \geq 4$, then vertices from K_p should be black.

Any efficient edge dominating set of a graph G (if it exists), is also a perfect edge dominating set of G.

Given a circular-arc graph $G = (V, E)$, we show how to solve MWPED in linear time. First, solve MWEED in $O(n)$ time. If there is a solution we save the one with minimum weight as a candidate solution. Therefore the candidates that should be explored are the perfect edge dominating sets that are not EED. Note that the set E is also a candidate solution.

We obtain circular-arc model $\mathcal{M} = (C, \mathcal{A})$ of G. We first show how to solve MWPED for G for the special cases where two or three arcs covers the entire circle. Then we show how to solve it for the remaining case.

5.6.1 A set of two or three arcs cover the entire circle

(a) There are 2 arcs, $A_v = (s_v, t_v), A_w = (s_w, t_w) \in \mathcal{A}$ such that $A_v \cup A_w = C$. Let $E' \neq E$ a perfect edge dominating set that is not a DIM. It is clear that E' determines a 3-coloring of the vertices from V that verifies (P1) and (P2). Let v and w be the corresponding vertices to A_v and A_w. The following possibilities form a valid color combination of v and w in order to satisfy (P1) and (P2):

(black,black). It is clear that any other arc A_z from the model that corresponds to the vertex $z \in V$ has common intersection with A_v or A_w.

Using (P2), z is not white. Hence, there is no white vertex in V. Therefore, $B = V$ and $E = E'$, absurd. This combination is not valid.

(black,gray) and (gray,black). Since both are symmetric, we consider one of them: (black,gray). The vertex w is gray and must verify (P1). The only non-white neighbor is v, hence the rest of his neighborhood should be white. By (P2) the rest of neighbors of v should not be white. Then there are no common vertices of v and w. Then for each vertex we know the color it must have, since every arc A_z intersects with either A_v or A_w. If there is no conflict with the colors each vertex have and verifies (P1) and (P2), we can save this solution as one more candidate solution.

(gray,gray). Every vertex z (different from v and w) should be adjacent to v and/or w. Since v and w are gray adjacent vertices and verify (P1), z is a white vertex. Thus E' contains the edge vw only, thus E' is a DIM, which contradicts the election of E'. Therefore this color combination is not valid.

(gray,white) and (white,gray). The vertex v is gray and should verify (P1), hence it must have exactly one non-white neighbor u. On the other hand, vertices from $N(w)$ are gray because (P2). We analyze the following two cases:

1. $A_u \subset A_v$: Every neighbor of u is also neighbor of v, hence they are all white. Vertices that are not neighbors of v are neighbors of w since their corresponding arcs are contained at A_w and should induce a matching since they satisfy (P2). It is easy to see that E' of this

combination is exactly the edges from the induced matching plus the edge vu. Thus E' is a DIM, which contradicts the election of E'. As a consequence, this combination is invalid for this case.

2. $A_u \cap A_w \neq \emptyset$, Then u is a gray vertex (P2). By (P1) vertices from $N(u) \setminus \{v\}$ are white. Vertices from $N(w)$ should be gray (P2). Thus there are no black vertices in G. Therefore E' is a DIM. The combination is invalid for this case.

Both combinations are invalid.

According to the previous cases there are at most two extra solutions that should be compared to other candidate solutions. One of these solutions is given by taking $E' = E$, and the other is a DIM of minimum weight).

(b) It does not satisfy (a) and there are three arcs $A_v, A_w, A_z \in \mathcal{A}$ such that $A_v \cup A_w \cup A_z = C$. Let $E' \neq E$ a perfect edge dominating set such that is not a DIM. It is clear that E' determines a 3-coloring of V that verifies (P1) and (P2). Let v, w, z the corresponding vertices to A_v, A_w, A_z. The possible combinations of colors that satisfy (P1) and (P2) are:

(black,black,black). In this combination, every arc A_u should have non empty intersection with one of A_v, A_w, A_z, then u is not white. Thus $B = V$ and $E' = E$. Contradiction. This combination is not valid.

(white,gray,gray),(gray,white,gray),(gray,gray,white). These three combinations are symmetric so we analyze one of them. Assume u is a black vertex. Then $A_u \cap A_v = \emptyset$ by (P2). Also $A_u \cap A_w = \emptyset$ and $A_u \cap A_z = \emptyset$ since w and z are gray adjacent vertices that should satisfy (P1). Thus A_u cannot be anywhere and u is not a black vertex. Hence $B = \emptyset$. It is clear that E' is a DIM and therefore this combination is invalid.

None of these combinations is valid, hence the best solution is among the candidates $E = E'$ and DIM of minimum weight.

(c) It does not satisfy (a) nor (b), in this case $\mathcal{M}$ is a Helly circular-arc model. This case is detailed in 5.6.2.

5.6.2 No set of two or three arcs cover the entire circle C

In this case G is Helly circular-arc graph and $\mathcal{M}$ a Helly circular-arc model of G. We compute $min_{p\in C}|\mathcal{A}(p)|$ and $max_{p\in C}|\mathcal{A}(p)|$ in order to analyze the different subcases:

(I) $min_{p\in C}|\mathcal{A}(p)| = 0$**:** Then $\mathcal{M}$ in an interval model and G an interval graph, hence a chordal graph. In [80] there is a linear time algorithm for MWPED for chordal graphs.

(II) $max_{p\in C}|\mathcal{A}(p)| = q \geq 4$**:** Thus there is a K_q formed by vertices that correspond to arcs that belong to $\mathcal{A}(p)$, where $|\mathcal{A}(p)| = q$. Let $v_1, \ldots, v_q$ be the vertices and $A_{v_1} = (s_{v_1}, t_{v_1}), \ldots, A_{v_q} = (s_{v_q}, t_{v_q})$ the corresponding arcs. The vertices $v_1, \ldots, v_q$ must be black vertices for any 3-coloring that corresponds to a perfect edge dominating set. We can replace each arc $A_{v_i}, 1 \leq i \leq q$ with two arcs $A_{v_i^-} = (s_{v_i}, p - \epsilon)$ and $A_{v_i^+} = (p + \epsilon, t_{v_i})$. Clearly, the resulting model $\mathcal{M}^*$ is an interval model because the point p does not belong to any arc. The intersection graph $G^* = (V^*, E^*)$ of $\mathcal{M}^*$ is an interval graph where each vertex $v_i, 1 \leq i \leq q$ is replaced by two non-adjacent vertices v_i^- and v_i^+ and the neighbors of v_i in G are either neighbors of v_i^- or neighbors of v_i^+ in G^* since $\mathcal{M}$ does not contains two arcs that cover the entire circle C. In addition, the vertices $v_1^-, \ldots, v_q^-$ induce a K_q in G^* and the vertices $v_1^+, \ldots, v_q^+$ induce another K_q in G^*. We can change the weight function ω to ω^* in G^* for each $vw \in E$, in the following way:

$$\omega^*(vw) = \begin{cases} \omega(vw) & v, w \in V \setminus \{v_1, \ldots, v_q\} \\ \omega(v_i w) & w \in V \setminus \{v_1, \ldots, v_q\}_\wedge v = v_i^-, 1 \leq i \leq q \\ \omega(v_i w) & w \in V \setminus \{v_1, \ldots, v_q\}_\wedge v = v_i^+, 1 \leq i \leq q \\ \omega(v_i v_j) & v = v_i^-{}_\wedge w = v_j^-, 1 \leq i, j \leq q \\ 0 & \text{otherwise} \end{cases}$$

We will show that each perfect edge dominating set E' of G corresponds to a perfect edge dominating set E'' of G^* with the same weight. Thus, the problem is reduced to solve MWPED for the interval graph G^* with the linear time algorithm [80].

Given a perfect edge dominating set E' of G, we show how to generate E''. Let the 3-coloring of G corresponding to E', where $v_1, \ldots, v_k$ are black vertices. In G^* those vertices are replaced by $v_1^-, \ldots, v_k^-$ and $v_1^+, \ldots, v_k^+$. The new 3-coloring for G^* consists on assigning black color to vertices $v_1^-, \ldots, v_k^-$ and $v_1^+, \ldots, v_k^+$, while the original vertices maintain the same color. This coloring must satisfy (P1) and (P2) in G^*. Otherwise, assume (P1) is not satisfied by a vertex w. Clearly, $w \in G$ and verifies (P1). If white and gray vertices as their adjacencies remain in both graphs, then $w \in G^*$ contains exactly one black vertex v_i, $1 \leq i \leq q$ in G, and in G^* it has two black neighbors v_i^- and v_i^+. This is a contradiction since in this case $A_w \cup A_{v_i}$ covers the entire circle C in $\mathcal{M}$. Thus (P1) is satisfied. On the other hand, every white vertex $w \in G$ verifies (P2). The vertices from $N(w)$ are gray and $N(w)$ remains the same at G^*, hence it satisfies (P2). Thus the 3-coloring of G^* corresponds to a perfect edge dominating set E'' from G^*. The sets E' and E'' have the same weight.

Given a perfect edge dominating set E'' from G^*, there is a corresponding perfect edge dominating set E' of G with the same weight.

Clearly, in the 3-coloring corresponding to E'', every vertex $v_1^-, \ldots, v_q^-$ and $v_1^+, \ldots, v_q^+$ is black because it is part of K_q with $q \geq 4$. In order to generate

the 3-coloring for G, assign $v_1, \ldots, v_q$ as black vertices and let the remaining vertices with the same color. It can be verified in a similar way that this 3-coloring verifies (P1) and (P2) in G, hence it corresponds to a perfect edge dominating set E' from G. Moreover the map of E' and E'' is the same as the previous one.

(III) $max_{p \in C}|\mathcal{A}(p)| = 3$**:** This implies G contains a K_3 formed by the vertices corresponding to $\mathcal{A}(p)$ where $|\mathcal{A}(p)| = 3$.

Let v_1, v_2, v_3 and their corresponding arcs $A_{v_1} = (s_{v_1}, t_{v_1}), A_{v_2} = (s_{v_2}, t_{v_2}), A_{v_3} = (s_{v_3}, t_{v_3})$. For every 3-coloring of G that satisfy (P1) and (P2), those three vertices must be: (i) three black vertices or (ii) exactly one of them white and the other two gray. We add an arc $(p - 2 * \epsilon, p + 2 * \epsilon)$ to the model, and call it $\mathcal{M}^+ = (C, \mathcal{A}^+)$ and G^+ as his corresponding graph.

It is easy to check that the 3-colorings of G that satisfy (P1), (P2) and **(i)** have a one-to-one correspondence with 3-colorings of G^+ that satisfy (P1) and (P2) since v_1, v_2, v_3 and the new vertex s corresponding to the additional arc are black vertices since those four vertices form a K_4 and s is not adjacent to any other vertex from G^+. In order to preserve the weight between the perfect edge dominating sets among G and G^+, the new edges v_1s, v_2s, and v_3s must have weight 0. Thus, MWPED can be solved for G^+ where the model $\mathcal{M}^+$ does not contain a set of two or three arcs that covers the entire circle C, is not an interval model and $max_{p \in C}|\mathcal{A}^+(p)| = 4$, which can be solved using the previous case in linear time. An additional consideration for 3-colorings of G that satisfy (P1), (P2) and not (i) must be added.

For the 3-colorings that satisfy (P1), (P2) but not (i), each arc $A_{v_j}, 1 \leq i \leq 3$ can be replaced by two arcs $A_{v_i^-} = (s_{v_i}, p - \epsilon)$ and $A_{v_i^+} = (p + \epsilon, t_{v_i})$ just as in (II), but instead of $q \geq 4$ arcs, now is just for 3 arcs. Again, the model $\mathcal{M}^*$ is an interval model and the intersection graph $G^* = (V^*, E^*)$ is an interval graph

where each vertex $v_i, 1 \leq i \leq 3$ is replaced by two non-adjacent vertices v_i^- and v_i^+.

We define 3 different weighted functions $\omega_j, 1 \leq j \leq 3$. For each edge $vw \in E^*$:

$$\omega_j(vw) = \begin{cases} \omega(vw) & v, w \in V \setminus \{v_1, \ldots, v_3\} \\ \omega(v_i w) & w \in V \setminus \{v_1, \ldots, v_3\} \wedge v = v_i^-, 1 \leq i \leq 3 \\ \omega(v_i w) & w \in V \setminus \{v_1, \ldots, v_3\} \wedge v = v_i^+, 1 \leq i \leq 3 \\ \omega(v_k v_l) & v = v_k^- \wedge w = v_l^- \wedge \{k, l\} = \{1, 2, 3\} \setminus \{j\} \\ 0 & v = v_k^+ \wedge w = v_l^+ \wedge \{k, l\} = \{1, 2, 3\} \setminus \{j\} \\ \infty & \text{otherwise} \end{cases}$$

It can be easily checked that any perfect edge dominating set E' from G has a corresponding 3-coloring that satisfies (ii).

A 3-coloring of G^* can be obtained from the previous one, by keeping the same colors for common vertices among both graphs, and v_i^-, v_i^+ get colors from v_i, $1 \leq i \leq 3$. The 3-coloring obtained corresponds to a perfect edge dominating set E'' from G^*. If the 3-coloring of G contains the vertex v_j, $j \in \{1, 2, 3\}$ with color white then the weight of E' using ω is the same weight that E'' using ω_j. Thus, the MWPED problem can be solved three times for the interval graph G^*, each time with a different function ω_j, $1 \leq j \leq 3$, applying linear algorithm from [31]. If the best of these solutions has bounded weight then the solution forms a perfect edge dominating set of minimum weight in G that satisfies (ii). Otherwise, there is no perfect edge dominating set of G whose three-coloring satisfies (ii). The minimum perfect edge dominating set of G is the minimum returned from the solutions given by [31] for G^* , the solution from G^+, E and the MWEED for G. All of them can be obtained in linear time.

(IV) $max_{p \in C}|\mathcal{A}(p)| = 2$ **and** $min_{p \in C}|\mathcal{A}(p)| \geq 1$**:** If $max_{p \in C}|\mathcal{A}(p)| > min_{p \in C}|\mathcal{A}(p)|$, then $min_{p \in C}|\mathcal{A}(p)| = 1$. In this case, we can find the induced cycle C_k with

$k \geq 4$ of G such that the arcs corresponding to the vertices covers the circle C in $\mathcal{M}$ and the rest of arcs from $\mathcal{A}$ are pairwise disjoint and are contained in exactly one of the arcs from C_k. The vertices from C_k are $v_1, v_2, \ldots, v_k$, $v_i v_{i+1} \in E$, $1 \leq i \leq k-1$, and $v_k v_1 \in E$, and the corresponding arcs of $v_1, v_2, \ldots, v_k$ are $A_1 = (s_1, t_1), A_2 = (s_2, t_2), \ldots, (s_k, t_k)$ and the circular order according to the extremes is: $s_1, t_k, s_2, t_1, s_3, t_2, \ldots, s_k$.

We analyze the following cases:

1. G is C_j, thus $k = n$. In this case $L(G)$ is exactly G. Instead of a weighted function over the arcs, the weighted function is over the vertices. (Note that C_n is a circular-arc graph)

2. There exists v_i with leaves. Without loss of generality, we assume $i = 2$ and $w_1, \ldots, w_{d(v_2)-2}$ are their leaves, $d(v_2) \geq 3$ is the degree of v_2. In this case, the vertex v_2 is father of w_1, in any other 3-coloring corresponding to a perfect edge dominating set of G, v_2 is not white since otherwise w_1 would be gray but could not have a non white neighbor to satisfy (P1). Then v_2 must be black or gray.

   (a) v_2 is black: Then $N(v_2)$ cannot be all white, so $w_1, \ldots, w_{d(v_2)-2}$ cannot be white. But at the same time this set cannot be gray because they are adjacent only to v_2, and has no white vertices to satisfy (P1). Then $w_1, \ldots, w_{d(v_2)-2}$ are black vertices. To solve this subcase, we can alterate the model in the following way: Add two identical arcs (t_1, s_3). In this new model, any leaf w_j, v_2 and the two new corresponding vertices to the additional arcs form a K_4, thus in any 3-coloring, these vertices must be black. But these two new vertices are adjacent only to v_2 and his leaves. Hence the remaining vertices can have any coloring independent of these new vertices. Therefore the perfect edge dominating sets from G in this subcase corresponds one-to-one to the modified model. The modified model does not contain

a set of 2 or 3 arcs that cover the circle C and there is a point p such that is contained in 4 different arcs. Algorithm (II) can be used to solve it. By assigning weight 0 to the added arcs then the solutions can be mapped and the weight will be the same.

(b) v_2 is gray: Then it contains exactly one non-white neighbor which must be one of $v_1, v_3, w_1, \ldots, w_{d(v_2)-2}$, and the rest are white vertices. Assume w_i is the non-white vertex, then it is black because there is no other neighbor, thus this 3-coloring satisfies (P1) and (P2). A new 3-coloring can be obtained, swapping color of w_i with the color of another leaf w_j. The new coloring will satisfy properties (P1) and (P2), hence it is convenient to choose the leaf w_j such that $\omega(v_2w_j)$ is minimum. Therefore there are 3 candidates to be considered as the non-white neighbor of v_2. These candidates are v_1, v_3 and the best w_j. For each case we may alter the model, adding an arc that intersects only to A_2 and the arc corresponding to the non-white vertex chosen. The new added edges have weight ∞ in order to prevent that in the new model the added vertex is chosen as the non-white vertex of v_2. This forces to choose the same non-white vertex from the original model. The new vertex has no other neighbors, hence it does not affect the coloring of the rest of the vertices. The resulting model has a point p which is contained by 3 different arcs, then the linear time algorithm (III) can be used to solve MWPED on this new model. The algorithm must be run 3 times, once for each election of the non-white vertex for v_2.

It applies at most once the algorithm from (II) and three time the algorithm from (III), hence the algorithm time complexity is $O(n + m)$.

Chapter 6

Cluster Vertex Deletion

The NP-Hard Cluster Vertex Deletion problem (CVD) asks for the minimum number of vertices to delete such that the resulting graph is a disjoint union of cliques, also known as a *Cluster* graph. We give polynomial algorithms for the problem on four different variants (weighted, unweighted, and with or without fixed number of cliques) for proper interval graphs, interval graphs, circular-arc graphs, permutation graphs and trapezoid graphs. We also show the complexity of the problem restricted to split graphs and bipartite graphs.

6.1 Preliminaries

Several clustering problems consist of identify similar objects, and exclude objects that do not clearly belong to any category. Those elements may be the result of measure errors, or simply outliers of the set. The goal is to identify those elements and remove them, in order to obtain the clusters of similar objects. There are several applications for clustering elements according to their pairwise relationships [4, 5, 45]. In terms of graph theory, each object is represented by a vertex, and two vertices are connected by an edge if and only if they are related as objects. The goal is to remove the minimum number of vertices in order to obtain a cluster graph.

We say G is *cluster* graph if and only if it is a disjoint union of complete graphs, or equivalently, a P_3-free graph. Unless is stated otherwise, a disjoint union of cliques means a P_3-free graph. The weighted CVD consists of finding the set of vertices with minimum weight such that their deletion transforms a given graph into a cluster graph. Formally:

Instance: A graph $G = (V, E)$, a vertex function $\omega : V \to [1, \infty)$, and $k \in \mathbb{N}$
Question: Is there a subset $X \subseteq V$ with $\sum_{v \in X} \omega(v) \leq k$ such that deleting all vertices in X from G results in a *cluster* graph ?

This problem will be denoted as *WCVD*. The unweighted version that asks whether exists a subset $X \subseteq V$ such that $|X| \leq k$ (all vertices have weight exactly one) is denoted as *UCVD*. There is an additional variant of these problems where an extra parameter D is given, for which the problem becomes whether $G = (V, E)$ can be divided into exactly D non-empty disjoint cliques. No edges are allowed between clusters. We denote these problems as *WDCVD* and *UDCVD* respectively. We say a cluster graph with exactly D cliques is a D-cluster graph. Note that $UCVD \leq_p UDCVD$ and $WCVD \leq_p WDCVD$. It is also easy to see that $UCVD \leq_p WCVD$ and $UDCVD \leq_p WDCVD$. We study the problems when the graphs are restricted to certain families.

6.2 Previous results

The *Cluster-Vertex-Deletion* problem has already been studied [13, 17] in terms of parameterized algorithms.

This problem is a variant of one of the most well studied problems, which is the *Cluster-Editing* problem, also known as *Correlation-Clustering*, which consists of edge editions (i.e. additions and deletions are allowed) in order to obtain a cluster graph. The literature for this problem is extensive [10, 11, 12, 26, 42, 43, 44, 64].

The *Cluster-Vertex-Deletion* problem can be stated as the minimum hitting set problem, where the set consists of the induced P_3s of the graph.

6.3 Algorithms and Complexity

In this section, we show the complexity of the problems when restricted to several graph classes. Moreover, whenever we show that a polynomial-time algorithm exists, we give a detailed algorithm. The algorithms are efficient from a practical point of view (i.e. no big constants hidden, and low degree polynomials) .

6.3.1 Interval graphs

Definition 6.1. *Define an interval I_i as two real numbers (s_i, t_i), where $s_i < t_i$. We say s_i is the start-point and t_i is the end-point.*

Note that any interval graph where two intervals share a point can be modified in order to avoid this sharing by shrinking one of the points an ϵ. The variable ϵ should be small enough to avoid any changes in the original adjacencies. Given that points are real numbers, this is always possible. We use the notation I_i instead of v_i in order to make reference to the vertex v_i represented by interval I_i.

Definition 6.2. *Let $\sigma = \{I_1, \ldots, I_n\}$ be an order of the vertices of an interval graph $G = (V, E)$ such that if $v_i < v_j$ then $s_i < s_j$.*

Definition 6.3. *Denote with $ND(i)$ the number j such that: $s_j > t_i$ and s_j is minimum.*

Let $M = \{I_1, \ldots, I_n\}$ be an interval graph model. Say S is a solution from M if S is the set of arcs that remains in M after the deletion of vertices (i.e. S is a cluster graph).

Lemma 6.1. *The values of $ND(i), \forall i : 1 \leq i \leq n$ can be computed in $O(1)$ time with an $O(n)$ preprocessing time for intervals graphs.*

Proof. Such an algorithm can be achieved by traversing the extreme-points in order from smallest to the largest (recall that points are represented by different real numbers). Whenever we find an end-point t_i we add it to a *pending* list, and whenever a start-point s_i is found, we retrieve elements of the *pending* list and we assign the index of current s_i (i) as the ND value for each element. Since each extreme-point is added and retrieved at most once from the *pending* list, it is trivial to see that the temporal and spatial complexity of the algorithm is $O(n)$ for preprocessing. □

Definition 6.4. *Let $mc_\sigma(i,j)$ is:*

1. *If $(v_i, v_j) \notin E(G)$ then $\emptyset$.*
2. *Otherwise: a maximum clique in $G[\{v_i, \ldots, v_j\}]$ according to σ-order, such that if $I_i, I_j \in mc_\sigma(i,j)$ and if $I_k \in mc_\sigma(i,j)$ then $t_k < t_j$.*

Definition 6.5. *Denote with t_{mc} the time to compute $mc_\sigma(i,j)$ from G, for all i,j such that $(v_i, v_j) \in E$.*

Use the analogous $mwc_\sigma(i,j)$ and t_{mwc} for the maximum weighted clique.

Lemma 6.2. *Given an interval graph sorted by the given σ-order, there is an algorithm for WDCVD problem with temporal complexity $O(D \cdot (n+m) + t_{mwc})$.*

Proof. We denote with $s_{i,D}(G)$ the sum of weights after the optimal deletions to convert $G[V_i]$ in a D-cluster graph, where $V_i = \{v_i, \ldots, v_n\}$ according to σ-order.

$$s_{i,D}(G) = \begin{cases} max\{s_{i+1,D}(G), max_{i \leq j \wedge (v_i,v_j) \in E}(s_{ND(j),D-1}(G) + mwc_\sigma(i,j))\} & i \leq n \\ 0 & i > n \ , \ D = 0 \\ \infty & i > n \ , \ D \neq 0 \end{cases}$$

The recursive function follows from the definition of values $s_{i,D}(G)$. The optimal way to delete vertices from V_i to form a D-cluster is to either (i) delete vertex i, and form a D-cluster with V_{i+1}, or to (ii) use from v_i to v_j as a cluster.

In case (i), no weights are added to the solution, since the problem gets reduced to solve the same instance without vertex v_i. In case (ii), the weight of the maximum weighted clique in $G[v_i, \ldots, v_j]$ is added to the result, since this clique is fixed as one of the cliques of the resultant cluster graph. The intuitive idea is that the recursion separates in two cases, either v_i is the first vertex (according to the order given) in the resultant graph, or it is deleted. In case it is deleted, the case (i) handles it by solving a smaller instance without v_i. Otherwise, the case is handled by fixing a clique that starts at v_i and ends at some later vertex v_j. Therefore, all possibilities are explored.

The optimal value of WDCVD is given by $s_{0,D}(G)$. Given that t_{mwc} is already computed and saved on a lookup table, the cost of the algorithm is $O(D \cdot (n+m))$ since there are $D \cdot n$ different values, and the function makes at most $d(v_i)$ recursive calls for each possible value i. □

Note that using the same formulation without the parameter D, leads to an algorithm for WCVD problem. Moreover, the same formulation using $mc_\sigma(i,j)$, leads to an algorithm for the unweighted versions of the problem (both UCVD, and UDCVD).

Corollary 6.1. *Given an interval graph sorted by the given σ-order, there is an algorithm for WCVD problem with temporal complexity $O(n+m+t_{mwc})$*

The cost of t_{mwc} in interval graphs is given by the computation of the maximum weighted cliques over all intervals i, j such that $I_i \in N(I_j)$. Recall that $mwc(i,j)$ is the value of the maximum weighted clique that uses I_i and I_j on the graph induced by $\{I_i, I_{i+1}, \ldots, I_j\}$ and such that there is no $I_k : t_k > t_j$. The following algorithm can do the job in $O(n+m)$ time. In order to simplify details, we assume we have the intervals sorted by s_i and for each interval we can traverse his neighborhood in increasing order of their extreme point t. The idea of the algorithm is to traverse, for each interval s_i, each interval $I_j \in N(s_i)$ in ascending order of t in order to compute each $mwc(i,j)$ value.

Algorithm 3 MWC(Interval Model M)

```
Assume M = {I_1, ..., I_n} sorted by s_i
counter := 0
for i = 1 to n do
  for all point p > s_i such that Interval(p) ∈ N[I_i] (ascending order of p) do
    if p is a start-point s_j then
      weight := weight + ω(I_j)
    else if p is an end-point t_j then
      mwc(i,j) := weight
      weight := weight - ω(I_j)
    end if
  end for
end for
```

It is easy to see that this algorithm has $O(n+m)$ time and space complexity.

Corollary 6.2. *There is an $O(D \cdot (n+m))$ algorithm for WDCVD problem and an $O(n+m)$ algorithm for WCVD problem*

Proper-Interval graphs

A *proper interval* graph is an interval graph that has an intersection model in which no interval properly contains another. In this section we give an $O(n)$ algorithm for the WCVD problem, given a model sorted by a σ-order. It can be naturally extended to an $O(n \cdot D)$ for the WDCVD problem.

Lemma 6.3. *Given two adjacent intervals $1 \leq i, j \leq n$ of a proper-interval model M, the value $mwc_\sigma(i,j)$ can be queried in $O(1)$ time using $O(n)$ preprocessing time.*

Proof. Assume w.l.o.g. $i < j$, and i, j represent the indexes of intervals according to σ-order. Recall that σ is the order of the start points in M. Since M is a proper-interval model, then $i < j \Rightarrow s_i < s_j$ and $t_i < t_j$. Since intervals I_i and I_j are

adjacent, then $t_i > s_j$. Note that for each k such that $i < k < j$ we have $s_i < s_k < s_j$, thus $t_i > s_k$. Therefore for each k such that $i < k < j$: $\{(v_k, v_i), (v_k, v_j)\} \in E$. The maximum weighted clique is then given by all vertices from $G[v_i, \ldots, v_j]$. This result can be obtained by precomputing a lookup table $Wsum[i]$ which indicates the sum of weights of the vertices from $G[v_1, \ldots, v_i]$. Note that $Wsum[j] = 0, j < 1$. The value $mwc_\sigma(i, j) = Wsum[j] - Wsum[i] + weight(v_i)$. The computation for $Wsum[]$ in $O(n)$ time is trivial. □

The previous results set the framework in order to give an algorithm for the WCVD problem on proper-interval graphs and prove it.

Algorithm 4 CVD-PI(Proper Interval Model M)

Assume $M = \{I_1, \ldots, I_n\}$ sorted by s_i
Compute $Wsum[]$ and $ND(i), \forall i$
$res[n] \leftarrow weight(I_n)$, $value[n] = Wsum[n]$, $last = n$
$S = \{(value[n], n)\}$
for $i = n - 1$ to 1 **do**
  $res[i] \leftarrow res[i + 1]$
  $value[i] = Wsum[i] + res[ND(i)]$
  $S = S \cup \{X_i = (value[i], i)\}$
  while $last >= i$ and $s_{last} > t_i$ **do**
    $S = S \setminus X_{last}$
    $last = last - 1$
  end while
  $res[i] = max(res[i], S_{max} - Wsum[i - 1])$, *where S_{max} is the maximum element over S*
end for
return $res[1]$

The algorithm iterates the intervals (sorted by s_i) backwards. Two results are computed, one where the current interval I_i is not in the resultant cluster graph, hence the solution is the same as the previous one where this interval was not processed

($res[i] \leftarrow res[i+1]$). The other one where this interval is part of the resultant cluster graph, in which case the next clique of the resultant graph should start at least from point $ND(i)$. Thus, the result for this case is computed as $WSum[i] + res[ND(i)]$ where $res[ND(i)]$ is an already optimal computed solution for the graph starting at interval $ND(i)$. The ***While*** loop deletes elements from S that can not be part of a clique within the current interval. At the end, the best result is given by the maximum between the case where interval i is not used, and the case where it is used, in which case the result is given by S_{max} (the maximum element over S, which represents the sum of weights of interval I_j such that $s_j \geq s_i$ and form a clique within interval I_i along with weights of intervals k such that $s_k < s_j$, minus the weight of elements that finishes before s_i.

Lemma 6.4. *Given a proper-interval model M, the algorithm returns the weight of the resultant graph after the optimal deletions to convert G in a D-cluster graph.*

Proof. We proceed with an inductive proof on the number of intervals. The base case is trivial. Let's assume the algorithm is correct for an n proper-interval model M_n. Let M_{n+1} be a proper-interval model of $n+1$ intervals. Assume w.l.o.g. $M_{n+1} = I_0 \cup M_n$.

The algorithm will make one more iteration. First, $res[0] \leftarrow res[1]$. In case an optimal solution without interval I_0 exists, by inductive hypothesis, $res[1]$ is the solution, since it represents the optimal solution for the graph $G[v_1, \ldots, v_n]$. The value of $value[i]$ is the sum of weights of intervals: $\{I_0, \ldots, I_i\}$ plus the optimal solution from $\{ND(i), \ldots, I_n\}$. This value is added to set S. Note that if I_0 belongs to an optimal solution O^*, then there exists an interval $I_k, k \geq 0$ such that $\{I_0, \ldots, I_k\}$ form a complete subgraph. The **while** loop finishes when it finds $s_last > t_0$, hence it finishes before reaching interval I_k, since $s_k < t_0$. Also S should contain value $value[k]$, and $Wsum[0-1]$ is 0 by definition. Thus, $res[i]$ is updated by the maximum value between $res[i+1]$ and $value[k] - WSum[-1]$ which is $WSum[k] - 0 + res[ND(k)]$.

□

Lemma 6.5. *Given a proper-interval model M, the algorithm runs in $O(n)$ time.*

Proof. A straightforward amortized analysis shows that the algorithm runs in $O(n)$ time if the operations over S are $O(1)$. We prove then that operations on the set S costs $O(1)$ amortized time. We propose a sorted linked-list for data representation. Note that values are given by the interval index and the value it represents that index. For deleting an element it is only necessary the index of the interval associated to the value to be deleted. Thus, deleting elements can be done in $O(1)$, using an additional structure of pointers that maps each interval to each value. The maximum can be obtained in $O(1)$ time in a sorted linked-list. The addition is the tricky operation. Let v_i be the value associated to interval I_i. The value v_i will be deleted after any value v_{i+1}, because the order of deletions depends on the order of iteration of variable *last*. Thus all values less than the value v_i can be deleted when v_i is inserted since we are only interested in the maximum of the set, and it will never be a smaller element than v_i. Thus each addition can also make deletions, but each element can be removed at most once. Since each interval is also added (and removed) at most once, the complexity over all operations over set S is $O(n)$. □

6.3.2 Circular-arc graphs

Recall that a *circular arc* graph is the intersection graph of a set of arcs on a circle where each vertex is represented as an arc, and two vertices are adjacent if and only if the corresponding arcs intersect. Note that interval graphs are a special case of circular-arc graphs.

Definition 6.6. *Define an arc A_i as two real numbers (s_i, t_i), where $s_i < t_i$. We say s_i is the start-point and t_i is the end-point according to clock-wise order.*

We say an *extreme-point* is either a start-point or an end-point of an arc. Unless otherwise stated, we will assume the circular model given has n arcs clockwise ordered.

It is easy to see that if no arc traverses a particular point p on the circle, then the graph can be represented as an interval graph. This can be done by defining a line

that starts at $p + \epsilon$ and ends at $p - \epsilon$, for sufficiently small ϵ, following the clockwise order from the original circle.

We proceed to show how to solve the WDCVD problem for circular-arc graphs, and then adapt it for the remaining variants of the problem. We assume $D > 1$, otherwise the problem can be solved using an existent algorithm for the maximum weighted clique problem on circular-arc graphs [91].

For solving the problem, it would be helpful to avoid sets of less than 4 arcs that cover the circle.

For a given point p, let $\mathcal{A}(p)$ the collection of arcs of that contains p. Let p be an extreme-point and $\mathcal{A}(p)$ such that $|\mathcal{A}(p)|$ is minimum among all points on the circle. As noted above, if $|\mathcal{A}(p)| = 0$ then the circular model can be converted to an interval model and the problem can be solved with the algorithm from the previous section. Otherwise, assume $|\mathcal{A}(p)| > 0$ and let S be the resulting cluster graph.

Case 1

No arc from S traverses p: If this is the case then by deleting all arcs traversing p we obtain a new interval model. It is clear that the optimal solution on this new model is the optimal solution for the original graph since we did not delete any of the arcs from the optimal solution.

Case 2

There is at least one arc from S traversing p: Thus there is a clique from S such that at least one of the arcs traverses p. Let C be this clique and define the point a as the last extreme-point of the clique C that you will find if you traverse it in anti-clockwise order and the point c as the last extreme-point of the clique C_i that you will find if you traverse it in clockwise order. Define $A_p = A_1, \ldots, A_k$, the set of arcs that traverse p.

Lemma 6.6. *At least one of the points a, c is an extreme-point from the set of arcs A_p.*

Proof. Suppose a, c are points from arcs outside A_p. Let a be a point from an arc B such that $B \notin A_p$. Therefore c cannot be given by B, otherwise B will traverse p or C will cover the entire circle. Since $D > 1$ this is not possible. Let c be a point from an arc B' such that $B' \notin A_p$. Clearly B' cannot traverse p neither. However it belongs to C, therefore it must be adjacent to B. Thus the clique C covers the entire circle. Absurd, since we assume $D > 1$. □

From the lemma above we know that at least one of the points $\{a, c\}$ is given by an extreme-point from the set A_p. It can be easily shown that if C belongs to S then no arc from S traverse points $a - \epsilon$ and $c + \epsilon$. Thus the optimal solution is given by the interval model that remains after deleting all arcs that traverse either a or c.

Thus, the algorithm looks both cases and chooses the best result among those.

Algorithm 5 WDCVD(Circular-arc Model M)

if $D = 1$ **then**
  Use [91]
end if
Search the extreme-point p such that $|\mathcal{A}(p)|$ is minimum
Create interval model M' where $A_j : p \in A_j$ are deleted
$Sol \leftarrow CVD - IntervalGraphs(M')$
for all $A_i : p \in A_i$ **do**
  Create interval model M' where $A_j : A_j \cap \{(s_i - \epsilon)\} \neq \emptyset$ are deleted
  $Sol_1 \leftarrow CVD - IntervalGraphs(M')$
  Create interval model M'' where $A_j : A_j \cap \{(t_i + \epsilon)\} \neq \emptyset$ are deleted
  $Sol_2 \leftarrow CVD - IntervalGraphs(M'')$
  $Sol \leftarrow BEST(Sol, Sol_1, Sol_2)$
end for
return Sol

The complexity of the algorithm is given by the complexity of the algorithm $CVD - IntervalGraphs$. Note that the bottleneck for the WDCVD algorithm for interval-graphs is given by t_{mwc}, which can be precomputed at the beginning of the algorithm. We can precompute t_{mwc} for the circular-arc graphs within the same

time, since we are only interested in computing maximum weighted clique of a set of arcs that do not cover the entire circle, this is, from an interval model. Let δ be the minimum degree vertex from the original circular-arc graph. By choosing an extreme-point from a lowest degree interval, the algorithm complexity is $O(\delta \cdot D \cdot (n+m))$ For the unweighted version an analogous algorithm can be done in $O(\delta \cdot (n+m))$.

Corollary 6.3. *There is an $O(\delta \cdot D \cdot (n+m))$ algorithm for WDCVD problem and an $O(\delta \cdot (n+m))$ algorithm for WCVD problem on circular-arc graphs*

Note that $\delta \leq \frac{m}{n}$, and the algorithm for proper-interval graph has $O(n)$ complexity, thus we state the following result:

Corollary 6.4. *There is an $O(D \cdot m)$ algorithm for WDCVD problem and an $O(m)$ algorithm for WCVD problem on proper-circular-arc graphs*

6.3.3 Permutation graphs

Let Π be a permutation over the set $S_n = \{1, 2, \ldots, n\}$. We define the n-vertex undirected graph $G(\Pi)$ with vertex set $V(G(\Pi)) = S_n$ and such that $(i, j) \in E(G(\Pi))$ if $i < j$ and $\Pi^{-1}(i) > \Pi^{-1}(j)$. A graph G is a permutation graph if exists a permutation Π such that G is isomorphic to $G(\Pi)$. We will make some abuse notation by calling indistinctly the vertices and the numbers in the permutation that represents those vertices.

A permutation graph can be represented by points in a 2-dimensional plane where each vertex is represented by a point [53]. Say a vertex v is represented as the point (v_x, v_y) in the plane. Then two vertices $(v, w) \in E(G)$ if and only if $v_x < w_x$ and $v_y > w_y$.

Definition 6.7. *Let C be a clique from G. We define $s(C)$ the point from C with minimum x-coordinate and $e(C)$ the point with maximum x-coordinate*

Note that for any clique C, y-coordinate from $s(C)$ is higher than y-coordinate from $e(C)$. Moreover, for any two disjoint cliques C_1, C_2 if $s(C_1)_x < s(C_2)_x$ then

- $e(C_1)_x \leq s(C_2)_x$
- $s(C_1)_y \leq e(C_2)_y$

The following figure describes the idea. The vertices that belong to the red part of the plane are in the neighborhood of p_1 but not of p_r, while vertices in the blue part of the plane are in the neighborhood of p_r but not p_1. The vertices in the purple part belongs to the neighborhood of both, p_1 and p_r.

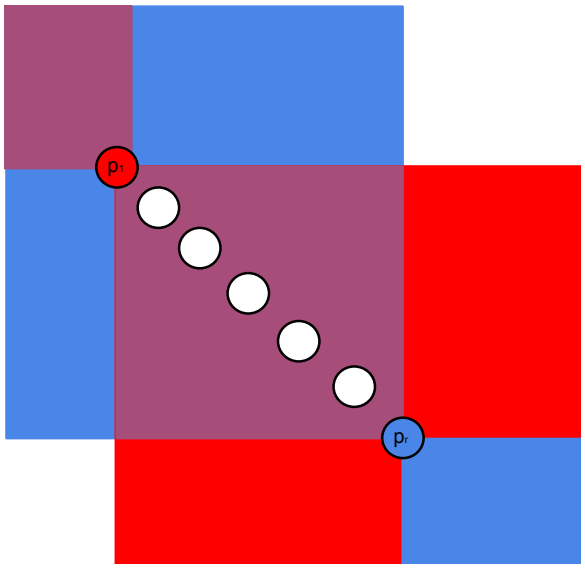

Figure 6.1: Red space contains neighbors of p_1, blue space contains neighbors of p_2, and purple space contains neighbors of p_1 AND p_r.

Two sets of points sorted by x-coordinate, $P = \{p_1, \ldots, p_r\}$ and $Q = \{q_1, \ldots, q_s\}$, such that each set is a complete, from two disjoint completes if and only if the following conditions are satisfied:

- $(p_r)_x \leq (q_1)_x$
- $(p_1)_y \leq (q_s)_y$

See the following figure where the purple boxes represents the complete sets, and it shows that the described properties are the natural way to separate boxes (i.e. complete sets).

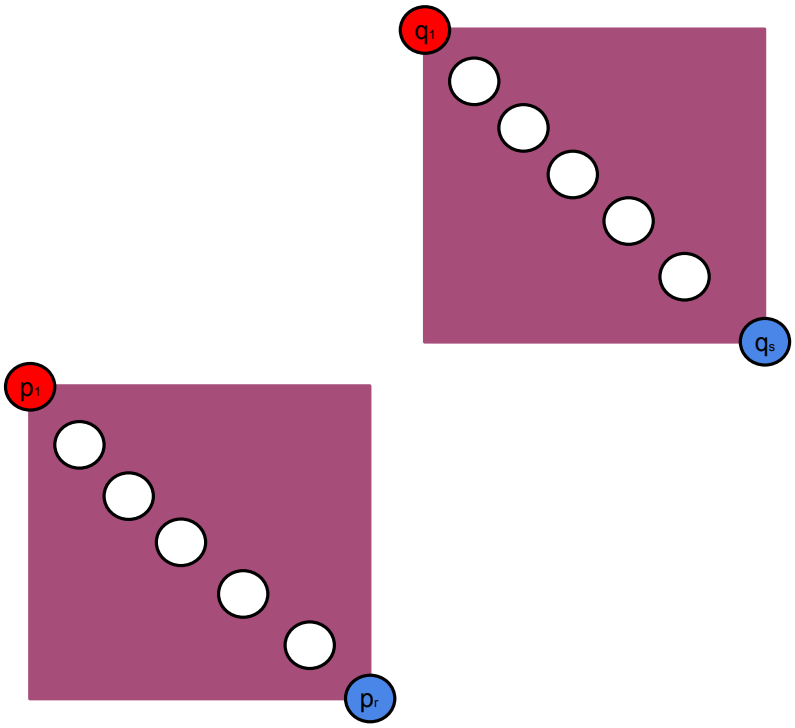

Figure 6.2: Two disjoint complete sets represented as disjoint boxes

.

Thus, given an arbitrary complete graph P as above, we can define the points $((p_1)_x, (p_r)_y)$ and $((p_r)_x, (p_1)_y)$ as the lower-left and upper-right bound of a box with sides parallel to x and y axis. Moreover two boxes are independent if and only if the two represented cliques are disjoint, this is, if upper-right corner of one box is lower and to the left of the lower-left corner of the second box. This representation is exactly the box-representation of trapezoid graphs [53]. Thus, each set of vertices that form a complete graph forms a box, and a maximum weighted independent set is a set of maximum weighted disjoint cliques in the original graph. Given n boxes, there is an $O(n \lg \lg n)$ time algorithm to resolve this problem over box-representation [53].

Algorithm 6 $PWCVD$(Permutation Model M)

Sort points by x-coordinate
$B \leftarrow \emptyset$
for $p = 1$ to n **do**
  $S \leftarrow < p_y, weight(p) >$
  for all point $q \in N(p)$ **do**
    $b =< (p_x, q_y), (q_x, p_y) >$ *//new box b*
    $B \leftarrow B \cup b$ *//Add b to box-representation*
    $E \leftarrow \{e : e \in S \wedge e_y > q_y\}$ *//Gets elements above q*
    $r \leftarrow e \in E$ such that $\forall e' \in E : e_y \leq e'_y$ *// Get lowest-element from E*
    $weight(b) \leftarrow weight(q) + weight(r)$ *//Set weight of box b*
    $S \leftarrow S \cup < q_y, weight(b) >$ *//Add box b to set S*
    $W \leftarrow \{e : e \in S \wedge e_y < q_y \wedge weight(e) < weight(b)\}$
    $S \leftarrow S \setminus W$
  end for
end for
return $MWIS(B)$ [53]

The idea of the algorithm is to form a box b for each pair of points, compute the weight of the maximum clique inside b, ans assign it to $weight(b)$, hence at the end the maximum-weighted-independent-set algorithm for boxes is executed. In order to compute the weight of each box, the algorithm iterates from lowest to highest x-coordinate. Thus, for each point p, we traverse, in ascending order, points q such that $q_x > p_x$, in order to obtain all the boxes with left-upper corner p. The new box b is represented by their corners and added to the box representation B. For each processed box, we maintain a point with their lowest-right point and his weight associated. The computation of the weight of the new box b is given by extending the maximum weighted box with point q. Boxes that can be extended with point q should have their lower-right corner above q, thus we look for it in the set E. Then we look

for the lowest element (minimum y-coordinate) of this set, named r. This element r is associated with a weight of a box that has its right-lower corner at r. Moreover, any other box with right-lower corner y-coordinate bigger than r has smaller weight, and should not be represented in E. This invariant is maintained by grouping those elements in W and then deleting from the set S, from where E gets their elements.

Lemma 6.7. *Algorithm PWCVD resolves the WCVD problem for permutation graphs.*

Proof. The proof follows from the previous idea. Each optimal solution can be represented by a set of independent boxes, where each possible box b is added to a new model B. Each box can be represented by a pair of adjacent vertices from the original graph. It is easy to see that the weight of the box b should be the maximum weight clique of points inside b. The set S contains elements that represents the best weight of a box already processed, given by the y-coordinate, and such that has its upper-right corner at point p. Thus the best box that has its lowest-right corner at a point q is given by the maximum weight of the box who is above and to the left of q plus the weight of q. Boxes W with lower-right corner below and to the left of q can be deleted from the set S since any other point q' that can be inside a box of W can also be inside the new box b. Therefore it is never an optimal decision to use a box represented by elements from W. Note that these deletions ensures that the new formed box b at each step should be extended from the first element above q. Thus, the resultant new model B contain a box for every possible complete that we are interested in. □

Lemma 6.8. *Algorithm PWCVD has time complexity* $O(m \lg n)$ *and space complexity* $O(n + m)$.

Proof. The set S can be implemented using a balanced tree sorted by y-coordinate. Given an element q_y, obtaining the first element e such that $e_y > q_y$ can be achieved in $O(\lg d)$ where d represents the number of nodes in the tree. Insertions and deletions

can also be done in $O(\lg d)$. Obtaining the set W and deleting is $O(\lg d + k)$ where k represents the size of W. Note that each element will be inserted and deleted at most once. Therefore the amortized complexity of deleting from W is $O(d \lg d)$. Since this operations are done for each closed neighborhood of each point, then we can bound the time complexity by $\sum_{i=1}^{n} d(v_i) \cdot \lg d(v_i)$. Note that the number of boxes B at the end is bounded by m, thus $MWIS(B)$ at the end of the algorithm can be done in $O(m \lg \lg m) \in O(m \lg n)$. Thus an upper bound for the time complexity for the algorithm is $O(m \lg n)$. □

For the WDCVD case, the problem that should be solved is the maximum-weighted-independent set with exactly D sets for the box-representation graph. Algorithm given in [53] can be easily extended for this problem resulting in a $O(D \cdot n \lg \lg n)$ time complexity algorithm. Thus the same idea can be used, and at the end, use the modified algorithm.

6.3.4 Trapezoid graphs

Trapezoid graphs are intersection graphs of trapezoids between two horizontal lines. Using the box-representation mentioned above, we can solve the WDCVD in polynomial time for this graph class. The idea to solve the problem on this graph family is an extension from the previous approach. Let $S = \{S_1, \ldots, S_D\}$ an optimal set of D disjoint cliques for the WDCVD problem. It is easy to see that S_i is a set of boxes that can be represented by a new box B such that the four corners of B are given by corner of boxes from S_i. For instance, the lower-left corner of box B is given by the coordinates b_x, b_y where b_x is the most left x-coordinate of S_i and b_y is the lowest y-coordinate of S_i. Since S_i is a complete subgraph, then (b_x, b_y) will belong to the same box. Therefore, for each possible set of four corners, we define a new box and add it to a new box-representation model. The number of boxes for the new model is bounded by $O(n^4)$. The weight of each box b is given by the maximum weighted clique in the original model such that is contained inside box b. Obtaining

the maximum weighted clique from each box can be done in $O(d \lg \lg d)$ where d is the number of boxes inside box b. Thus the cost for creating the new model is bounded by $\sum_{i=1}^{n^4} d(b_i) \lg \lg d(b_i) \in O(n^5 \lg \lg n)$. Applying the trapezoid algorithm on this new model gives us an $O(n^4 \lg \lg n)$ time algorithm for obtaining the optimal solution. Therefore the WDCVD problem can be solved in trapezoid graphs in polynomial time.

6.3.5 Split graphs

We show that *WCVD* for split graphs can be solved in polynomial time while *UDCVD* is NP-Complete. This result indicates that the parameter D is the barrier for the problem between being in P and in NPC.

Definition 6.8. *A split graph $G = (V, E)$ can be partitioned as $V = K \cup S$ where K is a maximal complete graph and S an independent set.*

Definition 6.9. *Define a weight function $w : V(G) \to R^+$ and for each set $S \subseteq V(G)$ denote $w(S) = \sum_{s_i \in S} w(s_i)$.*

Let G be a weighted split graph and $U \subset V(G)$ be an optimal set of vertices to delete from G. We assume G is connected since any isolated vertex v does not belong to any optimal set U. Note that by deleting either set K or S from G, we get a *cluster* graph. Hence $min\{w(K), w(S)\}$ is the cost of a solution given by deleting one of such sets, but it could exists better candidates (i.e. with minimum weight cost).
Let $K' = K \setminus U = \{k_1, \ldots, k_u\}$ and $S' = S \setminus U = \{s_1, \ldots, s_v\}$. Note that $w(U) < min\{w(S), w(K)\}$, otherwise an optimal solution can be found by deleting the set with minimum weight between K and S. In addition, it is easy to see that if $S \cup K$ forms one clique then $|S| = 1$ and as a consequence $|U| = 0$, hence the optimal solution is $U = \emptyset$ with $w(U) = 0$.

From now on we assume that $0 < w(U) < min\{w(S), w(K)\}$, since the solutions where $U = S$ or $U = K$ (hence $w(U) = min\{w(S), w(K)\}$ can be candidates to be checked at the end of the algorithm. Therefore $S' \neq \emptyset$ and $K' \neq \emptyset$.

Let $G' = \overline{K} \cup S$, where $V(G') = V(G)$ and $(v, w) \in E(G') \iff v \in V(K) \wedge w \in V(S)$. Recall that the maximum weighted independent set (MWIS) of a graph G is a set $S \subseteq V(G)$ such that S is independent and $w(S) \geq w(S')\ \forall S' \subseteq V(G)$, with S' an independent set.

Definition 6.10. KS-edges *:* $\{(u, v) \in E(G)\ s.t.\ u \in K\ \wedge\ v \in S\}$

We claim the following algorithm returns $\sum_{v \in U} w(u)$ (i.e. the weight of the vertices from the optimal *WCVD* solution):

Algorithm 7 WCVD(G: Split graph, w: Weight function)

$G' := \overline{K} \cup S$
$wU \leftarrow w(V) - w(MWIS(G'))$
for all $v \in S$ **do**
  $wU \leftarrow MIN(wU, w(V) - (w(MWIS(G'[N(v) \cup S \setminus v]) + w(v)))$
end for
return wU

We prove the following lemmas in order to show the correctness of the algorithm above.

Lemma 6.9. $|N(K') \cap S'| \leq 1$.

Proof. Suppose by contrary that $\exists s_1, s_2 \in S'$ such that both vertices has neighbors in K'. Since K' is a clique and S' an independent set then there is a path from s_1 to s_2 which has at least three vertices. But $K' \cup S'$ must be P_3-free. Hence we get a contradiction since $K' \cup S'$ is a solution by hypothesis. □

Lemma 6.10. *Any independent set of* $G' = \overline{K} \cup S$ *is an induced* cluster *graph in* G.

Proof. Suppose by contrary that exists an independent set I from G' such that I does not induce a *cluster* graph in G. Thus $\exists\{x_1, x_2, x_3\} \in I$ such that $\{(x_1, x_2), (x_2, x_3)\} \in E(G)$ and $(x_1, x_3) \notin E(G)$. Since $(x_1, x_3) \notin E(G)$, then at least one is not in K. Suppose w.l.o.g. $x_3 \in S$. Since $(x_2, x_3) \in E(G)$, one of them is in K, hence $x_2 \in K$. Therefore $\{x_1, x_2\} \in K$ and $x_3 \in S$. But since $(x_2, x_3) \in E(G)$ is a *KS-edge* then $(x_2, x_3) \in E(G')$, and therefore we get a contradiction since $\{x_1, x_2, x_3\}$ is not an independent set. □

Lemma 6.11.

Proof. It is trivial to see that an independent set of G' induces a cluster graph in G. Suppose by contrary that exists an induced *cluster* graph from G which is not an independent set of G'. Thus it contains at least one edge. If each edge contains a vertex from K' and one of S, then by lemma 6.9, all edges are connected to exactly one vertex $v \in S$ Hence the only solution is given by a cluster graph where there is exactly one clique C of size greater than one and v belongs C. Moreover, the only additional vertices that can be part of the resulting cluster graph are vertices from S such that no vertex is adjacent to a vertex from C, this is exactly an independent set of $N(v) \cup S \setminus \{v\}$. □

Corollary 6.5. *From lemmas 6.10, 6.11,* w(WCVD(G)) *is equivalent to* $w(V(G)) - w(MWIS(G'))$ *whenever no* KS-edges *are used.*

By lemma 6.9, all *KS-edges* of an induced *cluster* graph from G are adjacent to at most one vertex from S and if $v \in S$ belongs to the induced *cluster* graph, then $K' \cap \overline{N(v)} = \emptyset$. Thus it is possible to compute the best solution using each possible vertex $v \in S$ as the only vertex who has *KS-edges* adjacencies, by taking out this vertex from G' and also taking out $K \setminus N(v)$ since it has no common vertex with K'. Therefore the solution is the best (i.e. maximum) induced *cluster* graph from $N(v) \cup S \setminus v$ that does not use *KS-edges*.

Theorem 6.1. *CVD algorithm returns $w(U)$ for an optimal deletion set U and has polynomial running time*

Proof. By corollary 6.5, $w(MWIS(G'))$ at Step-1 is $w(U)$ if $|N(K') \cap S'| = 0$. Otherwise, using lemma 6.9 $|N(K') \cap S'| = 1$. At Step-3 algorithm handles this case by picking each possible vertex $v \in S$ such that $N(K') \cap S' = \{v\}$.
Now we can solve it as before by getting the $MWIS$ on the graph assuming there will be no other vertex $v' \in S'$. After fixing vertex v into the induced *cluster* graph, $K \setminus N(v)$ can be discarded, since $K' \cap \overline{N(v)} = \emptyset$. Any induced *cluster* graph in G will be an independent set in $G' \setminus v$ without *KS-edges* by lemma 6.11. But as v is within the induced *cluster* graph, no other *KS-edge* can be present too.

CVD algorithm has polynomial running time since G' (the graph after Step-1 is bipartite), and $MWIS(G)$ where G is bipartite can be solved in polynomial running time. □

UDCVD

Given the graph $G = (V, E)$ that can be easily partitioned into $V = K \cup S$ where K is a complete graph and S is an independent set, we need to find $X \subset V$ such that $V \setminus X$ is a D-cluster graph. We use the following known NP-Hard problem [96]:

Minimum Coverage Problem (MCP):

Instance: Two non-negative numbers k, D and a collection of sets $S = \{S_1, \ldots, S_m\}$ where $S_i \subseteq \{e_1, \ldots, e_n\}$

Objective: Does exists a subset $S' \subseteq S$ such that $|S'| = D$ and the number of covered elements $|\bigcup_{S_i \in S'} S_i| \leq k$?

Unweighted-D Cluster Vertex Deletion(UDCVD):

Instance: A graph $G = (V, E)$, a nonnegative number k and a positive integer D.
Objective: Does exists a subset $X \subseteq V$, such that deleting all vertices in X from G results in a cluster graph with exactly D cliques and $\|V \setminus X\| \geq k$?

Let us define $X' = V(G) \setminus X$, (i.e. vertices that remain in the resulting cluster graph). Obviously, as $|X|$ must be minimum, $|X'|$ must be maximum. Given a collection of sets $S = \{S_1, S_2, \ldots, S_m\}$ where $S_i \subseteq \{e_1, e_2, \ldots, e_n\}$ construct the following graph $G = (V, E)$:

- $V = V_e \cup V_s$ where $V_e = \{e_1, \ldots, e_n, e_{n+1}, e_{n+2}\}$ and $V_s = \{s_1, \ldots, s_m\} =$
- $E = \{(e_i, e_j) | 1 \leq i, j \leq n+2\} \cup \{(s_i, e_j) | e_j \in S_i\}$

Theorem 6.2. *The answer for $MCP(S, D, k)$ is yes if and only if the answer for $UDCVD(G, D+1, |D| + |V_e| - k)$ is yes, using the transformation given above.*

Proof. $\Rightarrow$) Let $MCP(S, D, k)$ be yes, and assume w.l.o.g. $S' = \{S_1, S_2, \ldots, S_D\} \subseteq S$ be a set such that $|\bigcup_{S_i \in S'} S_i| \leq k$. Let G be the graph using the transformation given above and let $X = V(G) \setminus \{N(s_1) \cup \ldots \cup N(s_D)\} \cup \{s_{D+1}, \ldots, s_m\}$. We proceed to show that $V(G) \setminus X$ results in a cluster graph with exactly $D+1$ cliques such that $|V \setminus X| \geq |D| + |V_e| - k$. Thus $UDCVD(G, D+1, |D| + |V_e| - k)$ is yes. Let $H = G \setminus X$. H contains the vertices $\{s_1, \ldots, s_D\}$. Since their neighbors were deleted they form an independent set of $|D|$ isolated vertices (i.e. cliques). Besides that, there are at least $|V_e| - k$ elements in $G \setminus X$, which are the vertices not adjacent to $\{s_1, \ldots, s_D\}$. Note that $\{e_{n+1}, e_{n+2}\}$ are not contained in any S_i element, thus $|V_e| - k > 0$. These elements form a clique in H since V_e is a complete set. Thus we have $D+1$ cliques formed by these D isolated vertices plus the clique from vertices in V_e. The number of elements is clearly at least $|D| + |V_e| - k$.

$\Leftarrow$) Let $UDCVD(G, D+1, |D|+|V_e|-k)$ be yes. It is easy to see that at most one clique from the resulting cluster graph will contain vertices from V_e. Also, any clique that contains (s_i, e_j) can be replaced by the bigger clique $C \setminus \{s_i\} \cup \{e_{n+1}, e_{n+2}\}$, where C is the clique containing the edge (s_i, e_j). Thus an optimal solution from $UDCVD(G, D+1, |D|+|V_e|-k)$ will contain a set S_D of D isolated vertices from the set V_s plus a subset of V_e. Therefore adjacent vertices of S_D are the only vertices that will not be in the resulting cluster graph, so we want to minimize $N(S_D) = k$. Thus the cluster graph has D vertices from S and $|V_e| - k$ vertices from V_e with k being minimum. These k vertices are covered by exactly D elements from V_s, hence those elements can be chosen as an optimal MCP solution. As a consequence $MCP(S, D, k)$ is yes. □

6.3.6 Bipartite graphs

A graph $G = (V, E)$ is *bipartite*, if its edge set E satisfies $E \subseteq \{ (v, v') \,|v \in V_1, v' \in V_2\}$ where $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$. Partitions of a *bipartite* graph can be easily obtained in $O(n + m)$ time.

UCVD

A subset of vertices in a graph is called a *dissociation set* if it induces a subgraph with a vertex degree of at most 1. The *maximum dissociation set problem* (i.e. the problem of finding a dissociation set of maximum size in a given graph) is NP-hard even for bipartite graphs with maximum degree 3 and C_4-free [14]. It is easy to see that for bipartite graphs the maximum dissociation set problem is equivalent to find the maximum number of disjoint K_1 and K_2. This is exactly the *UCVD*, since cliques in bipartite graphs are either K_1 or K_2. We know the problem is *NP-Hard* even for subclasses of the bipartite graphs, thus *UCVD* is *NP-Hard* for bipartite graphs, even when restricted to maximum degree 3, or C_4-free [14], or planar bipartite graphs [88].

We know *WCVD* can be solved using *WDCVD* and *UCVD* can be solved using *WCVD*, hence all the variants of CVD are *NP-Hard* for planar bipartite graphs.

Chapter 7

Conclusions

In this thesis, we have provided new results on several dominating-set related problems, and we have put additional effort on make these results relevant for practical purposes. Any algorithm from this thesis can be easily implemented, and the theoretical upper-bounds will match the expected computation time in a real implementation, since there are no big hidden polynomials, nor big hidden constants. Moreover, the algorithms were greatly simplified on purpose to make them more easy for the reader to understand the ideas.

At the beginning of the thesis, we show some interesting results on the theoretical side of the problem, by giving results on the problem for several graph classes restricted by forbidding induced subgraphs. This is useful to understand the right path to solve the problem whenever the input is restricted to some of the studied graph families. We showed that using available knowledge and a few more proofs chosen in the correct order, the gap missing for the complexity of many interesting graph families can be closed. These results were submitted to a journal.

The chapter where the Roman domination is addressed, gives a few interesting insights about relationship between problems. We showed that a naive approach for the dominating-set problem may set a framework to solve different variants, such as the Roman-dominating set problem. The advantage is the simplicity of the

algorithms, along with the fact that gives interesting bounds on the algorithms when restricted to graphs with certain properties, such as graphs that contain a dominating set of constant size. Moreover, these simple approaches were enough to improve results on the algorithms for certain graph classes. Finally, we give an extremely simple algorithm for dominating set in cographs. These results show that there is space for improvement even in known problems and well-studied graph classes, using simple ideas. These results were accepted to *RAIRO - Theoretical Informatics and Applications* journal.

The results on the Efficient-Edge-Domination (DIM) problem show that the problem can be approached in different ways, either exponential-time algorithms, or by posing restrictions to the input domain. The coloring-rules have shown to be a useful and elegant way to approach the problem, since it shows a natural way to partially solve the problem, and makes it evident the procedure to follow. The ideas that arise from this approach are easy to understand, even the proofs may be harder to follow since many cases must be analyzed. This approach may be useful for several problems where the decisions are based on the selection of edges, but can be simplified to coloring vertices, which in turn leads to simple ideas that may provide an efficient exponential time algorithm. The results for the problem restricted to different graph classes show that even smart algorithms can be improved, and sometimes, a simple bound may be enough to decrease the complexity of the algorithm. The main exponential algorithm was presented at *The 24th International Symposium on Algorithms and Computation (ISAAC 2013)* and the full paper submitted to a journal, while several results when restricted to certain graph classes was presented in *Latin American Theoretical INformatics Symposium (LATIN 2014)*, and a full version published in *Information Processing Letters* journal.

Additionally, we present results on the Perfect and Efficient Dominating set, for the vertex and edge cases, restricted to circular-arc graphs. We showed previous results and added new algorithms in order to understand these four problems for this graph class. The new results lead to efficient algorithms.

The Cluster-Vertex-Deletion problem was addressed, as far as we know, for the first time in terms of restricted graph families. We attack this problem for many of the most well known and studied graph classes, where it makes sense to solve clustering problems.

It is interesting to note that almost all of the developed algorithms are fast in the practical sense, even the exponential time ones have low polynomial and low constants associated with it, and are easy to code. Even no implementation is given as part of the thesis, several of the algorithms have been implemented and proved to be fast, it remains as part of future work to refactor codes and show them with a suitable test set, experimental data and its conclusions.

Results

Note that n represents the number of vertices of the given graph G and m the number of edges.

- A total completion of the complexity dichotomy (polynomial time vs NP-completeness) of minimum dominating set problem for graph families defined by forbidden induced subgraphs of size at most four. This includes several new results:
  - The problem can be solved in polynomial time for the following graph families: $4K_1$-free, $(2K_2, claw)$-free, $(2K_2, diamond)$-free, $(2K_2, co-claw)$-free, $(2K_2, paw)$-free.
  - The problem falls in the NP-Complete class for the following graph families: (K_3, C_4)-free, planar of maximum degree 3 and $(K_4, C_4, C_5, C_7, C_8, C_9, C_{10}, C_{11}, diamond, claw)$-free
- Results over domination and Roman-domination restricted P_5-free graphs:

- A robust algorithm to solve minimum dominating set problem restricted to (P_5,(s,t)-net)-free in $O(m)$ time if $s \leq 2$, $O(m^2)$ time if $s \leq 4$, and $O(mn^{s-3} + m^{\frac{s}{2}})$ otherwise.
- A robust algorithm to solve minimum Roman-dominating set problem restricted to (P_5,(s,t)-net)-free in $O(m)$ time if $s \leq 2$, $O(mn^2)$ time if $s \leq 4$, and $O(mn^{s-3} + m^{\frac{s}{2}})$ otherwise.
- A robust and extremely simple algorithm for both problems restricted P_4-free graphs that runs in $O(m)$ time.

- Results over weighted and counting version of the Efficient-Edge-Domination (DIM) problem:
  - An $O^*(min\{4^{|D|}, 1.7818^n\})$ time algorithm with linear space complexity,where $|D|$ is the size of a dominating set. Note that whenever $|D|$ is a constant, the algorithm runs in linear time on the graph size.
  - An $O(nm^2\mu)$ algorithm where μ is the number of maximal independent sets in the graph. This leads to an $O^*(1.44225^n)$ time algorithm for general graphs and $O^*(1.41421^n)$ for bipartite graphs. Note that whenever the number of maximal independent sets μ is bounded by a polynomial function, the algorithm runs in polynomial time.
  - An $O(1.1939^n m)$ time algorithm with linear space complexity.
  - Bounds to improve algorithms from $O(n + m)$ to $O(n)$ time for chordal, dually-chordal and biconvex graphs.
  - An improvement from $O(n^2)$ time algorithm for the unweighted-version problem restricted to claw-free graphs to an $O(n)$ algorithm for the weighted and counting version.

- Results over the efficient and perfect domination problem for vertices and edges restricted to circular-arc graphs

  - Polynomial time algorithms for the Minimum weighted efficient edge domination, the Minimum weighted perfect vertex domination and the minimum weighted perfect edge domination problems

- Results over the variants of Cluster-Vertex-Deletion problem:

  - An $O(n)$ time algorithm for the $WCVD$ problem and $O(n \cdot D)$ time for the $WDCVD$ problem, restricted to proper interval graphs.
  - An $O(n+m)$ time algorithm for the $WCVD$ problem and $O(D \cdot (n+m))$ time for the $WDCVD$ problem, restricted to interval graphs.
  - An $O(m)$ time algorithm for the $WCVD$ problem and $O(D \cdot m)$ time for the $WDCVD$ problem, restricted to circular-arc graphs.
  - An $O(m \lg n)$ time with $O(n+m)$ space complexity algorithm for the $WCVD$ problem and $O(D \cdot n \lg \lg n)$ for the $WDCVD$ problem, restricted to permutation graphs.
  - A polynomial-time algorithm for the $WDCVD$ problem, restricted to trapezoid graphs.
  - A polynomial-time algorithm for the $WCVD$ problem, and a proof that the $UDCVD$ problem is NP-Complete, when restricted to split graphs.
  - A proof that the $UCVD$ problem is NP-Complete, when restricted to bipartite graphs.

Bibliography

[1] Asratian, A. S., Denley, T. M. J., and Häggkvist, R. (1998). *Bipartite Graphs and Their Applications*. Cambridge University Press, New York, NY, USA. 2

[2] Bacso, G. and Tuza, Z. (1990). Dominating cliques in P_5-free graphs. *Periodica Mathematica Hungarica*, 21(4):303 – 308. 34

[3] Baker, K. A., Fishburn, P. C., and Roberts, F. S. (1972). Partial orders of dimension 2. *Networks*, 2(1):11–28. 2

[4] Bansal, N., Blum, A., and Chawla, S. (2004). Correlation clustering. *Machine Learning*, 56(1):89–113. x, 105

[5] Ben-Dor, A., Shamir, R., and Yakhini, Z. (1999). Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–297. x, 105

[6] Bertossi, A. A. (1984). Dominating sets for split and bipartite graphs. *Inf. Process. Lett.*, 19:37–40. iv, vi, 15, 17, 23, 26, 28

[7] Biggs, N., Lloyd, E. K., and Wilson, R. J. (1986). *Graph Theory, 1736-1936*. Clarendon Press, New York, NY, USA. 4

[8] Björklund, A. (2010). Determinant Sums for Undirected Hamiltonicity. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 173–182, Washington, DC, USA. IEEE Computer Society. 40

[9] Björklund, A., Husfeldt, T., Kaski, P., and Koivisto, M. (2007). Fourier meets möbius: fast subset convolution. In [66], pages 67–74. 40

[10] Böcker, S. (2012). A golden ratio parameterized algorithm for cluster editing. *J. Discrete Algorithms*, 16:79–89. 106

[11] Böcker, S., Briesemeister, S., Bui, Q. B. A., and Truß, A. (2009). Going weighted: Parameterized algorithms for cluster editing. *Theor. Comput. Sci.*, 410(52):5467–5480. 106

[12] Böcker, S., Briesemeister, S., and Klau, G. W. (2011). Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334. 106

[13] Böcker, S. and Damaschke, P. (2011). Even faster parameterized cluster deletion and cluster editing. *Inf. Process. Lett.*, 111(14):717–721. xi, 106

[14] Boliac, R., Cameron, K., and Lozin, V. V. (2004). On computing the dissociation number and the induced matching number of bipartite graphs. *Ars Comb.*, 72. 127

[15] Boliac, R. and Lozin, V. V. (2002). On the clique-width of graphs in hereditary classes. In *Proceedings of the 13th International Symposium on Algorithms and Computation*, ISAAC '02, pages 44–54, London, UK. Springer-Verlag. 16, 17, 26

[16] Booth, K. S. and Johnson, J. H. (1982). Dominating sets in chordal graphs. *SIAM J. Comput.*, 11(1):191–199.

[17] Boral, A., Cygan, M., Kociumaka, T., and Pilipczuk, M. (2013). Fast branching algorithm for cluster vertex deletion. *CoRR*, abs/1306.3877. xi, 106

[18] Brandstädt, A., Engelfriet, J., and Lozin, V. (2006). Clique-width for 4-vertex forbidden subgraphs. *Theory of Computing Systems*, 39(4):561–590. 27

[19] Brandstädt, A., Fičur, P., Leitert, A., and Milanič, M. (2015). Polynomial-time algorithms for weighted efficient domination problems in at-free graphs and dually chordal graphs. *Information Processing Letters*, 115(2):256–262. 41

[20] Brandstädt, A., Hundt, C., and Nevries, R. (2010). Efficient edge domination on hole-free graphs in polynomial time. In *Proceedings of the 9th Latin American conference on Theoretical Informatics*, LATIN'10, pages 650–661, Berlin, Heidelberg. Springer-Verlag. vii, 12, 40, 42, 45, 66, 69

[21] Brandstädt, A., Le, V. B., and Spinrad, J. P. (1999). *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA. 2, 4

[22] Brandstädt, A., Leitert, A., and Rautenbach, D. (2012). Efficient dominating and edge dominating sets for graphs and hypergraphs. *CoRR*, abs/1207.0953. vii, 12, 40, 41, 42, 67, 82

[23] Brandstädt, A. and Lozin, V. V. (2003). On the linear structure and clique-width of bipartite permutation graphs. *Ars Comb.*, 67. 42

[24] Brandstädt, A. and Mosca, R. (2011). Dominating induced matchings for P_7-free graphs in linear Time. *CoRR*, abs/1106.2772. vii, 12, 40, 42

[25] Calamoneri, T., Finocchi, I., and Italiano, G. F., editors (2006). *Algorithms and Complexity, 6th Italian Conference, CIAC 2006, Rome, Italy, May 29-31, 2006, Proceedings*, volume 3998 of *Lecture Notes in Computer Science*. Springer. 138

[26] Cao, Y. and Chen, J. (2012). Cluster editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169. 106

[27] Cardoso, D. M., Cerdeira, J. O., Delorme, C., and Silva, P. C. (2008). Efficient edge domination in regular graphs. *Discrete Applied Mathematics*, 156(15):3060–3065. vii, 12, 40, 42

[28] Cardoso, D. M., Korpelainen, N., and Lozin, V. V. (2011). On the complexity of the dominating induced matching problem in hereditary classes of graphs. *Discrete Appl. Math.*, 159(7):521–531. vii, 12, 40, 42, 70, 71, 73, 75, 76

[29] Cardoso, D. M. and Lozin, V. V. (2009). Dominating induced matchings. In [77], pages 77–86. vii, 12, 40, 41, 45, 49

[30] Chang, M. (1998). Efficient algorithms for the domination problems on interval and circular-arc graphs. *SIAM J. Comput.*, 27:1671–1694. 5, 82

[31] Chang, M. and Liu, Y. (1994). Polynomial algorithms for weighted perfect domination problems on interval and circular-arc graphs. *J. Inf. Sci. Eng.*, 11(4):549–568. 89, 91, 93, 102

[32] Chao, H. S., Hsu, F., and Lee, R. C. T. (2000). An optimal algorithm for finding the minimum cardinality dominating set on permutation graphs. *Discrete Applied Mathematics*, 102(3):159–173. iv, vi, 5, 15, 17, 26, 29

[33] Chen, Z. and Zhang, S. (2002). Tight upper bound on the number of edges in a bipartite $K_{3,3}$-free or K_5-free graph with an application. *Inf. Process. Lett.*, 84(3):141–145. 68

[34] Chiba, N. and Nishizeki, T. (1985). Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223. 34

[35] Cockayne, E. J., Jr., P. A. D., Hedetniemi, S. M., and Hedetniemi, S. T. (2004). Roman domination in graphs. *Discrete Mathematics*, 278(1-3):11 – 22. vi, 28, 29

[36] Cook, S. A. (1971). The complexity of theorem-proving procedures. In [62], pages 151–158. i, 1

[37] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition. ii, 3

[38] Courcelle, B., Makowsky, J., and Rotics, U. (1999). Linear time solvable optimization problems on graphs of bounded clique width. *Theory of Computing Systems*, 33:125–150. 16, 17, 26

[39] Dabrowski, K. K., Lozin, V. V., Raman, R., and Ries, B. (2012). Colouring vertices of triangle-free graphs without forests. *Discrete Math.*, 312(7):1372 – 1385. 16, 17

[40] Dahllöf, V. and Jonsson, P. (2002). An algorithm for counting maximum weighted independent sets and its applications. In [49], pages 292–298. 42

[41] Dahllöf, V., Jonsson, P., and Wahlström, M. (2005). Counting models for 2sat and 3sat formulae. *Theor. Comput. Sci.*, 332(1-3):265–291. 42

[42] Damaschke, P. (2005). On the fixed-parameter enumerability of cluster editing. In [71], pages 283–294. 106

[43] Damaschke, P. (2006). Fixed-parameter tractable generalizations of cluster editing. In [25], pages 344–355. 106

[44] Damaschke, P. (2013). Cluster editing with locally bounded modifications revisited. In [73], pages 433–437. 106

[45] Dehne, F. K., Langston, M. A., Luo, X., Pitre, S., Shaw, P., and Zhang, Y. (2006). The cluster editing problem: Implementations and experiments. In *IN PROC. 2ND IWPEC*, pages 13–24. Springer. x, 105

[46] Downey, R. G. and Fellows, M. R. (1999). *Parameterized Complexity*. Springer-Verlag. 530 pp. ii, 3

[47] Durán, G., Lin, M. C., Mera, S., and Szwarcfiter, J. (2008). Algorithms for finding clique-transversals of graphs. *Annals of Operations Research*, 157:37–45. 10.1007/s10479-007-0189-x. 80, 81

[48] Durán, G., Lin, M. C., Mera, S., and Szwarcfiter, J. L. (2006). Algorithms for clique-independent sets on subclasses of circular-arc graphs. *Discrete Applied Mathematics*, 154(13):1783–1790. 80, 81

[49] Eppstein, D., editor (2002). *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*. ACM/SIAM. 138

[50] Erdős, P., Goodman, A. W., and Pósa, L. (1966). The representation of a graph by set intersections. *Canadian Journal of Mathematics*, 18:106–112. ii, 2, 4

[51] Farber, M. and Keil, J. M. (1985). Domination in permutation graphs. *J. Algorithms*, 6(3):309–321. vi, 29

[52] Feige, U. (1998). A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652. 5

[53] Felsner, S., Mller, R., and Wernisch, L. (1997). Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Applied Mathematics*, 74(1):13 – 32. 116, 118, 119, 121

[54] Fiat, A. and Sanders, P., editors (2009). *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*. Springer. 143

[55] Fomin, F. and Kratsch, D. (2010). *Exact Exponential Algorithms*. Texts in theoretical computer science. Springer. 40, 56, 59, 60

[56] Fomin, F. V., Gaspers, S., Saurabh, S., and Stepanov, A. A. (2009). On two techniques of combining branching and treewidth. *Algorithmica*, 54(2):181–207. 42

[57] Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA. 5, 6, 7, 25

[58] Gavril, F. (1974). Algorithms on circular-arc graphs. *Networks*, 4(4):357–369. 2

[59] Gavril, F. and Yannakakis, M. (1980). Edge dominating sets in graphs. *SIAM J. Appl. Math.*, 38(3):364–372. 17

[60] Golumbic, M. and Laskar, R. (1993). Irredundancy in circular arc graphs. *Discrete Applied Mathematics*, 44(13):79 – 89. 83

[61] Grinstead, D. L., Slater, P. J., Sherwani, N. A., and Holmes, N. D. (1993). Efficient edge domination problems in graphs. *Inf. Process. Lett.*, 48(5):221–228. vii, 12, 40, 42

[62] Harrison, M. A., Banerji, R. B., and Ullman, J. D., editors (1971). *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*. ACM. 137

[63] Hedetniemi, S. and Laskar, R. (1990). Bibliography on domination in graphs and some basic definitions of domination parameters. *Discrete Mathematics*, 86(13):257 – 277. 6

[64] Hüffner, F., Komusiewicz, C., Liebtrau, A., and Niedermeier, R. (2014). Partitioning biological networks into highly connected clusters with maximum edge coverage. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 11(3):455–467. 106

[65] Johnson, D. S. (1973). Approximation algorithms for combinatorial problems. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 38–49, New York, NY, USA. ACM. 5

[66] Johnson, D. S. and Feige, U., editors (2007). *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*. ACM. 135

[67] Jünger, M., Reinelt, G., and Rinaldi, G., editors (2003). *Combinatorial Optimization - Eureka, You Shrink!, Papers Dedicated to Jack Edmonds, 5th International Workshop, Aussois, France, March 5-9, 2001, Revised Papers*, volume 2570 of *Lecture Notes in Computer Science*. Springer. 144

[68] Kloks, T., Kratsch, D., and Müller, H. (2000). Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121. 27

[69] Korpelainen, N. (2009). A polynomial-time algorithm for the dominating induced matching problem in the class of convex graphs. *Electronic Notes in Discrete Mathematics*, 32:133–140. vii, 12, 40, 42

[70] Kratsch, D. (2000). Domination and total domination on asteroidal triple-free graphs. *Discrete Applied Mathematics*, 99(1–3):111 – 123. iv, vi, 15, 17, 21, 26, 29, 37

[71] Kratsch, D., editor (2005). *Graph-Theoretic Concepts in Computer Science, 31st International Workshop, WG 2005, Metz, France, June 23-25, 2005, Revised Selected Papers*, volume 3787 of *Lecture Notes in Computer Science*. Springer. 138

[72] Land, A. H. and Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520. ii, 3

[73] Lecroq, T. and Mouchard, L., editors (2013). *Combinatorial Algorithms - 24th International Workshop, IWOCA 2013, Rouen, France, July 10-12, 2013, Revised Selected Papers*, volume 8288 of *Lecture Notes in Computer Science*. Springer. 138

[74] Lekkeikerker, C. and Boland, J. (1962). Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64. 2

[75] Liedloff, M., Kloks, T., Liu, J., and Peng, S. (2008). Efficient algorithms for roman domination on some classes of graphs. *Discrete Applied Mathematics*, 156(18):3400–3415. vi, 29, 30, 38, 39

[76] Lin, M. C., McConnell, R. M., Soulignac, F. J., and Szwarcfiter, J. L. (2008). On cliques of helly circular-arc graphs. *Electronic Notes in Discrete Mathematics*, 30:117–122. 86

[77] Lipshteyn, M., Levit, V. E., and McConnell, R. M., editors (2009). *Graph Theory, Computational Intelligence and Thought, Essays Dedicated to Martin Charles Golumbic on the Occasion of His 60th Birthday*, volume 5420 of *Lecture Notes in Computer Science*. Springer. 137

[78] Livingston, M. and Stout, Q. F. (1988). Distributing resources in hypercube computers. In *Proceedings of the third conference on Hypercube concurrent computers and applications: Architecture, software, computer systems, and general issues - Volume 1*, C3P, pages 222–231, New York, NY, USA. ACM. vii, 12, 40

[79] Lozin, V. V. and Milanič, M. (2006). Domination in graphs of low degree. *Rutcor Research Report (RRR) New Jersey 27*. 23

[80] Lu, C. L., Ko, M., and Tang, C. Y. (2002). Perfect edge domination and efficient edge domination in graphs. *Discrete Applied Mathematics*, 119(3):227–250. vii, 12, 40, 42, 67, 85, 86, 95, 99, 100

[81] Lu, C. L. and Tang, C. Y. (1998). Solving the weighted efficient edge domination problem on bipartite permutation graphs. *Discrete Applied Mathematics*, 87(1-3):203–211. vii, 12, 40, 42

[82] McConnell, R. M. (2003). Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147. 82

[83] McKee, T. A. (1999). Topics in intersection graph theory. 7

[84] Merris, R. (2003). Split graphs. *Eur. J. Comb.*, 24(4):413–430. 2

[85] Milanič, M. (2013). Hereditary efficiently dominatable graphs. *Journal of Graph Theory*, 73(4):400–424. 41

[86] Müller, H. and Brandstädt, A. (1987). The NP-completeness of steiner tree and dominating set for chordal bipartite graphs. *Theor. Comput. Sci.*, 53:257–265. 17

[87] Nicolai, F. and Szymczak, T. (2001). Homogeneous sets and domination: A linear time algorithm for distance - hereditary graphs. *Networks*, 37(3):117–128. vi, 21, 29

[88] Orlovich, Y. Dolgui, A., Finke, G., Gordon, V., and Werner, F. (2011). The complexity of dissociation set problems in graphs. *Discrete Applied Mathematics*, 159(13):1352 – 1366. 127

[89] Poghosyan, A. and of the West of England, U. (2010). *The Probabilistic Method for Upper Bounds in Domination Theory*. PhD thesis. University of the West of England. 4

[90] Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition. ii, 3

[91] Shih, W. and Hsu, W. (1989). An $O(nlogn + mloglogn)$ maximum weight clique algorithm for circular-arc graphs. *Information Processing Letters*, 31(3):129 – 134. 114, 115

[92] Spinrad, J. (2003). *Efficient Graph Representations.: The Fields Institute for Research in Mathematical Sciences*. Fields Institute monographs. American Mathematical Soc. 76, 83

[93] Tsukiyama, S., Ide, M., Ariyoshi, H., and Shirakawa, I. (1977). A new algorithm for generating all the maximal independent sets. *SIAM J. Comput.*, 6(3):505–517. 55

[94] van Rooij, J. M. M., Nederlof, J., and van Dijk, T. C. (2009). Inclusion/exclusion meets measure and conquer. In [54], pages 554–565. 42

[95] Vazirani, V. V. (2001). *Approximation Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA. ii, 3

[96] Vinterbo, S. A. (2007). A stab at approximating minimum subadditive join. In Dehne, F. K., Sack, J., and Zeh, N., editors, *WADS*, volume 4619 of *Lecture Notes in Computer Science*, pages 214–225. Springer. 125

[97] Wahlström, M. (2008). A tighter bound for counting max-weight solutions to 2sat instances. In *Proceedings of the 3rd international conference on Parameterized and exact computation*, IWPEC'08, pages 202–213, Berlin, Heidelberg. Springer-Verlag. 42

[98] Wikipedia (2014a). Bipartite graph — wikipedia, the free encyclopedia. [Online; accessed 20-November-2014]. 9

[99] Wikipedia (2014b). Circular-arc graph — wikipedia, the free encyclopedia. [Online; accessed 20-November-2014]. 10

[100] Wikipedia (2014c). Interval graph — wikipedia, the free encyclopedia. [Online; accessed 20-November-2014]. 8

[101] Wikipedia (2014d). Permutation graph — wikipedia, the free encyclopedia. [Online; accessed 20-November-2014]. 10

[102] Wikipedia (2014e). Split graph — wikipedia, the free encyclopedia. [Online; accessed 20-November-2014]. 11

[103] Woeginger, G. J. (2001). Exact algorithms for np-hard problems: A survey. In [67], pages 185–208. 40

[104] Yannakakis, M. (1982). The complexity of the partial order dimension problem. 17

[105] Zverovich, I. E. (2003). The domination number of (K_p, P_5)-free graphs. *Australasian Journal of Combinatorics*, 27:95–100. vi, 15, 26, 29