

Tesis Doctoral

Algoritmos aproximados para problemas de optimización combinatoria

Vasiliev, Saveliy

2018-03-21

Este documento forma parte de la colección de tesis doctorales y de maestría de la Biblioteca Central Dr. Luis Federico Leloir, disponible en digital.bl.fcen.uba.ar. Su utilización debe ser acompañada por la cita bibliográfica con reconocimiento de la fuente.

This document is part of the doctoral theses collection of the Central Library Dr. Luis Federico Leloir, available in digital.bl.fcen.uba.ar. It should be used accompanied by the corresponding citation acknowledging the source.

Cita tipo APA:

Vasiliev, Saveliy. (2018-03-21). Algoritmos aproximados para problemas de optimización combinatoria. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires.

Cita tipo Chicago:

Vasiliev, Saveliy. "Algoritmos aproximados para problemas de optimización combinatoria". Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 2018-03-21.

EXACTAS UBA

Facultad de Ciencias Exactas y Naturales



UBA

Universidad de Buenos Aires



UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Algoritmos aproximados para problemas de optimización combinatoria

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

Saveliy Vasiliev

Director de tesis: Dr. Julián Diego Mestre

Director asistente: Dr. Min Chih Lin

Consejero de estudios: Dr. Javier Marengo

Lugar de trabajo: Instituto de Cálculo, FCEyN, Universidad de Buenos Aires

Buenos Aires, 2018.

Fecha de defensa: 21 de Marzo de 2018

To my parents

Contents

I	Resúmenes en Español	iii
	Resumen	iv
	Introducción	v
	Transversal de cliques	viii
	Conjunto clique independiente	ix
	Recubrimiento por vecindades cerradas	x
	Conjunto vecindad independiente	xi
	Matching inducido	xii
	Conclusiones	xiii
II	Thesis	xv
	Abstract	xvi
	Agradecimientos	xvii
1	Introduction	2
2	Induced Matching	26
3	Clique Transversal Set	37
4	Clique Independent Set	48
5	Neighborhood Covering Set	58
6	Neighborhood Independent Set	72
7	Conclusions and further work	92

Parte I

Resúmenes en Español

Resumen

Algoritmos aproximados para problemas de optimización combinatoria

En esta tesis estudiamos algoritmos aproximados para cinco problemas de optimización combinatoria. Exploramos los casos pesados y no pesados para diferentes clases de grafos donde estos problemas son NP-completos.

Un *transversal de cliques* de un grafo G es un conjunto de vértices que interseca a todas las cliques maximales de G . El problema de optimización asociado es encontrar un transversal de cliques mínimo.

Un *conjunto clique independiente* de un grafo G es un conjunto de cliques maximales de G que no comparten vértices entre sí. El problema de optimización asociado es encontrar un conjunto clique independiente máximo.

Para estos dos problemas presentamos algoritmos aproximados para grafos de línea, grafos con grado acotado, y grafos planares.

Un *recubrimiento por vecindades cerradas* de un grafo G es un conjunto de vértices tal que toda arista del grafo pertenece a la vecindad cerrada de alguno de estos vértices. El objetivo es encontrar un recubrimiento por vecindades cerradas mínimo. Para este problema presentamos un algoritmo con factor arbitrariamente bueno para grafos planares.

Un *conjunto vecindad independiente* de un grafo G es un conjunto de aristas tal que la vecindad cerrada de cualquier vértice de G contiene a lo sumo una de estas aristas. El objetivo es encontrar un conjunto vecindad independiente máximo. Para este problema damos algoritmos aproximados para grafos sin diamantes y grafos planares. Además mostramos un algoritmo polinomial para cografos.

Un *matching inducido* de un grafo G es un matching tal que ninguna arista de G conecta a dos aristas del matching. El objetivo es encontrar el matching inducido máximo. Presentamos un algoritmo $\Delta(G)$ -aproximado para la versión pesada de este problema, donde $\Delta(G)$ es el grado máximo de G . Además mostramos algoritmos aproximados para grafos sin $K_{1,k}$ y algunas subclases de estos.

Palabras clave. Algoritmos aproximados, optimización combinatoria, transversal de cliques, conjunto clique independiente, recubrimiento por vecindades cerradas, conjunto vecindad independiente, matching inducido.

Introducción

Un gran número de problemas de optimización que surgen de la actividad humana tienen una naturaleza discreta. Es decir, uno ha de encontrar una solución óptima en un conjunto finito de posibles soluciones. Estos problemas se llaman *problemas de optimización combinatoria*. Su naturaleza implica que mucha de la maquinaria matemática desarrollada para optimizar funciones sobre el dominio de los reales y complejos no sea de mucha utilidad para estos problemas. En la mayoría de los casos prácticos no parece ser viable encontrar soluciones analíticamente, lo que sugiere el uso de computadoras para buscar soluciones a estos problemas.

Al tomar este enfoque nos enfrentamos con las limitaciones de poder de cómputo: muchos de los problemas de optimización combinatoria son NP-difíciles [41]. Si bien al día de la fecha no se sabe si existen algoritmos eficientes para resolver tales problemas, en la comunidad científica hay un amplio consenso sobre la tesis que sostiene que no existen; y si llegasen a existir, se cree que serían muy poco útiles desde el punto de vista práctico. Sin embargo, esto no nos exime de la necesidad de resolver estos problemas. Esto justifica el desarrollo de algoritmos para problemas NP-difíciles que no cumplan con todas las propiedades deseables: ser eficientes, computar una solución óptima, y funcionar en cualquier instancia válida de input. El sacrificio de cada una de estas propiedades nos introduce en amplios campos de investigación.

Si sacrificamos la eficiencia, es decir, la garantía de que el algoritmo termina en tiempo polinomial, estamos en presencia de varias ramas de estudio: los algoritmos de optimización poliedral [25, 76, 80], los algoritmos de fuerza bruta [26], los algoritmos *branch and reduce* y otros [37, 38, 89].

Si sacrificamos generalidad, estamos restringiendo el dominio de nuestro algoritmo. Por ejemplo, es NP-difícil encontrar una clique de tamaño máximo en un grafo general, sin embargo se puede hacer de forma eficiente para algunas familias de grafos, como los grafos de intervalo [43].

Si sacrificamos optimalidad, estamos en el área de las heurísticas polinomiales. Aquí encontramos algoritmos aleatorios¹ que funcionan “bien” con alta probabilidad [75]; los algoritmos que funcionan “bien” en la práctica [42, 83]; y los algoritmos cuya

¹Técnicamente también renunciamos al *determinismo* si hablamos de algoritmos aleatorios.

solución tiene cierta calidad garantizada por un argumento matemático [86, 88]. Estos últimos son los algoritmos aproximados.

Formalmente diremos que, para un problema de minimización, un algoritmo es α -aproximado si computa una solución factible de costo a lo sumo α veces el costo óptimo en tiempo polinomial. De manera similar, un algoritmo para un problema de maximización se dice α -aproximado si el valor de la solución computada es al menos una α -ésima parte del valor óptimo.

Para diseñar algoritmos aproximados es de suma importancia encontrar buenas cotas para el óptimo tales que éstas puedan ser relacionadas con alguna solución factible computada en tiempo polinomial. Esta relación es la que nos permite demostrar el factor de aproximación. Hay que enfatizar que estas cotas pueden ser de carácter teórico—es suficiente que uno pueda relacionar la solución del algoritmo con esta cota mediante algún factor multiplicativo.

Existen varias técnicas ampliamente utilizadas en el diseño de algoritmos aproximados. En este trabajo utilizaremos principalmente tres: algoritmos golosos, la técnica de Baker para grafos planares [3], y *fractional local ratio* [5, 20].

Los algoritmos golosos son aquellos algoritmos iterativos que buscan maximizar el beneficio en cada iteración. Las demostraciones de los factores de aproximación se basan muchas veces en relacionar directamente una solución óptima con la computada.

La técnica de Baker para grafos planares permite obtener algoritmos con factor $(1 \pm \epsilon)^2$ para cualquier $\epsilon > 0$ fijo [3]. Sin embargo, el tiempo de ejecución es exponencial en $1/\epsilon$, por lo cual estos algoritmos son poco útiles en la práctica. Aún así, permiten establecer que un problema es “fácil” en los grafos planares a pesar de ser NP-difícil. La idea de este método es dividir el grafo planar en varias “capas” de cierto grosor fijo (dependiente de ϵ). Cada capa de un grosor fijo tiene *treewidth* acotado, lo que muchas veces permite resolver el problema en cuestión en tiempo polinomial en cada capa. Si logramos argumentar que el error incurrido para “combinar” las soluciones a las capas en una solución global es “pequeño” relativo al grosor de las capas, podemos argumentar que el factor es arbitrariamente bueno.

Por último, *local ratio* es una técnica cuya idea central es encontrar una solución factible y una descomposición de los costos de forma que podamos acotar los errores incurridos por nuestra solución computada respecto a cada componente. Si

²Para factores arbitrariamente buenos utilizamos una convención diferente a la definida mas arriba: para los problemas de maximización diremos que un algoritmo es $(1 - \epsilon)$ -aproximado si el valor computado es al menos $(1 - \epsilon)$ veces el valor óptimo.

respecto de cada componente de la descomposición de costos nuestra solución es α -aproximada, entonces—bajo supuestos muy generales—la solución será α -aproximada para los costos originales. Aquí el punto crucial es encontrar una buena descomposición que nos permita probar un factor pequeño de aproximación. Para esto podemos utilizar información provista por una solución fraccionaria a un programa lineal. El uso de esta información en tal algoritmo se conoce como *fractional local ratio*.

En el [Capítulo 2](#) estudiamos el matching inducido máximo con pesos. En el [Capítulo 3](#) estudiamos el problema transversal de cliques mínimo. En el [Capítulo 4](#) estudiamos el problema relacionado: conjunto clique independiente máximo. Seguido a esto estudiamos otros dos problemas que se relacionan entre sí: en el [Capítulo 5](#) exploramos el recubrimiento de aristas por vecindades cerradas mínimo; y en el [Capítulo 6](#) el conjunto vecindad independiente máximo. Por último, las conclusiones y preguntas que dejamos abiertas están presentadas en el [Capítulo 7](#).

Transversal de cliques

Un subconjunto S de vértices de un grafo G es un *transversal de cliques* si S interseca a todas las cliques maximales de G . Hallar un transversal de cliques de tamaño mínimo es un problema NP-completo para grafos de línea, planares y de grado acotado [47].

En este capítulo analizamos el comportamiento de las cliques en los grafos de línea. Este análisis nos permite dar un algoritmo 3-aproximado para el caso no pesado y un 4-aproximado para el caso pesado.

Por otro lado, es fácil ver que existe un algoritmo 4-aproximado para grafos planares pesados. Para grafos planares sin pesos mejoramos este factor, dando una 3-aproximación. Siguiendo esta línea de investigación, adaptamos resultados ya conocidos para ver que existe un algoritmo lineal para grafos con treewidth acotado para el caso pesado. Esta observación nos sirve para derivar un algoritmo $(1 + \epsilon)$ -aproximado para grafos planares pesados para cualquier $\epsilon > 0$ fijo. Sin embargo, el tiempo de ejecución de este algoritmo es prohibitivo en la práctica, a diferencia de los algoritmos con factor 3 y 4 que damos antes.

Por último, si el grado máximo de G , $\Delta(G)$, está acotado por una constante fija, probamos que existe un algoritmo $\left\lceil \frac{\Delta(G)+1}{2} \right\rceil$ -aproximado para grafos no pesados.

Conjunto clique independiente

Un subconjunto de cliques maximales de G mutuamente disjuntas es un *conjunto clique independiente*. Hallar tal conjunto de tamaño máximo es NP-completo para grafos de línea, grafos planares y grafos de grado acotado [47]. En la variante pesada del problema los pesos de cada clique están dados por la suma de los pesos de los vértices de dicha clique.

En este capítulo damos un algoritmo lineal para grafos pesados con treewidth acotado, que luego utilizamos para dar un $(1 - \epsilon)$ -aproximado para grafos planares pesados para cualquier $\epsilon > 0$ fijo.

Además, damos un $\max\left\{3; \frac{\Delta(G)+3}{5} + \epsilon\right\}$ -aproximado para grafos de línea sin pesos con gado máximo $\Delta(G)$.

Recubrimiento por vecindades cerradas

Un subconjunto S de vértices de un grafo G es un *recubrimiento por vecindades cerradas* de G si toda arista de G pertenece a la vecindad cerrada de algún vértice en S . Hallar un recubrimiento por vecindades cerradas mínimo es un problema NP-completo para grafos planares [47].

En este capítulo mostramos que este problema es lineal para grafos pesados con treewidth acotado. Este resultado nos permite dar un algoritmo $(1 + \epsilon)$ -aproximado para grafos planares con vértices pesados.

Conjunto vecindad independiente

Un subconjunto de aristas S de un grafo G es un *conjunto vecindad independiente* si no existe un vértice de G que contenga dos aristas de S en su vecindad cerrada. Hallar un conjunto vecindad independiente máximo es un problema NP-completo para grafos planares [47].

En este capítulo damos un algoritmo goloso $\Delta(G)$ -aproximado para grafos pesados. Además, para grafos pesados sin diamantes como subgrafos inducidos exhibimos un algoritmo $(\Delta(G)/2 + 1)$ -aproximado.

Probamos que existe un algoritmo polinomial para grafos con treewidth acotado. Utilizamos este resultado para derivar un algoritmo $(1 - \epsilon)$ -aproximado para grafos planares pesados, para cualquier $\epsilon > 0$ fijo.

Mostramos además que existe un algoritmo polinomial para cografos pesados.

Por último, damos una cota superior asintóticamente ajustada de la cardinalidad de un conjunto vecindad independiente para grafos regulares.

Matching inducido

Un *matching inducido* de G es un matching tal que no existe una arista de G que conecte dos aristas de este matching. Dicho en otras palabras, es un matching de G tal que el subgrafo de G inducido por los vértices que toca el matching es 1-regular, o equivalentemente: es el mismo matching. Hallar un matching inducido de tamaño máximo es un problema NP-completo [81].

En varios trabajos se han estudiado algoritmos aproximados para este problema restringido a grafos regulares y subclases de estos [32, 46, 56, 78, 91]. Sin embargo, solo se conoce un algoritmo $2(\Delta(G) - 1)$ -aproximado para grafos con grado a lo sumo $\Delta(G)$ [91]. Nosotros presentamos un algoritmo $\Delta(G)$ -aproximado para el caso pesado de este problema.

Por último, exploramos el comportamiento de un algoritmo goloso en grafos pesados que no tengan $K_{1,k}$ como subgrafo inducido y subclases de estos.

Conclusiones

En esta tesis estudiamos cinco problemas de optimización combinatoria desde el punto de vista de su aproximabilidad.

En el 2000, Guruswami y Rangan [47] demostraron que el transversal de cliques mínimo, conjunto clique independiente máximo, recubrimiento por vecindades cerradas mínimo y conjunto vecindad independiente máximo son NP-completos para grafos planares. Nosotros presentamos algoritmos con factor de aproximación arbitrariamente bueno para cada uno de estos problemas en sus versiones pesadas. De esta forma contribuimos a la clasificación de la complejidad de estos problemas en los grafos planares: dentro de los NP-completos estos son los más sencillos. Para obtener estos resultados hemos dado algoritmos polinomiales para grafos pesados cuyo treewidth es acotado. Cabe destacar que estos resultados son de índole teórica, ya que en la práctica estos algoritmos tienen tiempos de ejecución muy altos.

Además de los resultados de NP-completitud en grafos planares, en [47] se ha probado que estos problemas son NP-completos para otras clases de grafos. Nosotros dimos algunos algoritmos aproximados para algunos problemas restringidos a estas clases. De esta forma, esta tesis contiene una respuesta parcial a [47] desde el área de algoritmos aproximados.

Por último, logramos bajar el factor $2(\Delta(G) - 1)$ de Zito [91] a $\Delta(G)$ para matching inducido pesado con grafos de grado a lo sumo $\Delta(G)$. Además dimos algunos resultados de aproximabilidad para grafos $K_{1,k}$ -free y subclases pesadas.

Lista de resultados obtenidos

- Para transversal de clique.
 - Una 3-aproximación para grafos de línea no pesados.
 - Una 4-aproximación para grafos de línea pesados.
 - Un algoritmo práctico 3-aproximado para grafos planares sin pesos.
 - Un algoritmo lineal para grafos pesados con treewidth acotado.
 - Un PTAS para grafos planares con pesos.

- Una $\left\lceil \frac{\Delta(G)+1}{2} \right\rceil$ -aproximación para grafos no pesados con $\Delta(G)$ acotado por una constante fija.
- Para conjunto clique independiente.
 - Un algoritmo lineal para grafos pesados con treewidth acotado.
 - Un PTAS para grafos planares pesados.
 - Una $\max \left\{ 3, \frac{\Delta(G)+3}{5} + \epsilon \right\}$ -aproximación para grafos de línea sin pesos.
- Para recubrimiento de vecindades cerradas.
 - Un algoritmo lineal para grafos pesados con treewidth acotado.
 - Un PTAS para grafos planares pesados.
- Para conjunto vecindad independiente.
 - Una $\Delta(G)$ -aproximación para grafos pesados.
 - Una $(\Delta(G)/2 + 1)$ -aproximación para grafos pesados sin diamantes.
 - Un algoritmo polinomial para cografos pesados.
 - Una cota asintóticamente ajustada de la cardinalidad de un conjunto vecindad independiente en grafos regulares.
 - Un algoritmo lineal para grafos pesados con treewidth acotado.
 - Un PTAS para grafos planares pesados.
- Para matching inducido.
 - Una $\Delta(G)$ -aproximación para grafos pesados.
 - Una $(2k - 3)$ -aproximación para grafos $K_{1,k}$ -free pesados.
 - Una k -aproximación para grafos $(K_{1,k}, \text{chair})$ -free pesados.

Part II

Thesis

Abstract

Approximation algorithms for combinatorial optimization problems

In this thesis we study approximation algorithms for five combinatorial optimization problems. We explore the weighted and the unweighted variations for different graph classes where these problems are NP-complete.

A *clique transversal set* of a graph G is a subset of vertices that intersects each maximal clique of G . The optimization problem is to find a maximum clique transversal set. For this problem we present approximation algorithms on line graphs, bounded degree graphs, and planar graphs.

A *clique independent set* of a graph G is a set of maximal cliques of G such that no two of them share a vertex. The optimization problem is to find a maximum clique independent set. For this problem we present approximation algorithms on line graphs and planar graphs.

A *neighborhood covering set* of a graph G is a subset of vertices such that each edge of G belongs to some closed neighborhood of these vertices. The optimization problem is to find a minimum neighborhood covering set. For this problem we give an arbitrary good approximation ratio for planar graphs.

A *neighborhood independent set* of a graph G is a subset of edges of G such that the closed neighborhood of any vertex in G contains at most one of these edges. The optimization problem is to find a maximum neighborhood independent set. For this problem we give approximation algorithms for bounded degree graphs, diamond-free graphs, and planar graphs. Furthermore, we give a polynomial time algorithm for cographs.

An *induced matching* of a graph G is a matching such that no edge of G connects two edges of this matching. The optimization problem is to find a maximum induced matching. For the weighted case we give a $\Delta(G)$ -approximation algorithm, where $\Delta(G)$ is the maximum degree of G . Furthermore, we give approximation algorithms for $K_{1,k}$ -free graphs and subclasses.

Keywords. Approximation algorithms, combinatorial optimization, clique transversal set, clique independent set, neighborhood covering set, neighborhood independent set, induced matching.

Agradecimientos

Primero quiero dar las gracias a mis directores, Julián Mestre y Min Chih Lin. Trabajar estos años bajo su dirección ha sido un placer y fue una experiencia muy enriquecedora, tanto en lo personal como en lo académico.

Asimismo quiero agradecer a mi consejero de estudios y mi director de tesis de licenciatura, Javier Marengo, quien me ha dado innumerables consejos durante muchos años que nos conocemos. Su capacidad de trabajo y dedicación son un verdadero ejemplo a seguir, además de su sincera y notable vocación al servicio de otros.

Muchísimas gracias a los jurados de esta tesis, Celina Miraglia Herrera de Figueiredo, Esteban Zindel Feuerstein y Luciano Norberto Grippo, por aceptar leer este documento y brindar sus comentarios y sugerencias que mejoraron la presentación del trabajo.

Al Instituto de Cálculo, al Departamento de Computación, a la Facultad de Ciencias Exactas y Naturales, a la Universidad de Buenos Aires y al Centro Nacional de Investigaciones Científicas y Técnicas. Agradezco a las personas que integran estas instituciones y a la sociedad Argentina que permite su existencia. Gracias. Gracias por abrirme las puertas siendo yo una persona de otro país.

Me alegra haber encontrado a tanta gente maravillosa en mi paso por la facultad, tanto en grado como posgrado. Es difícil, sino imposible, no olvidar a nadie intentando enumerarlos.

Los cursos dictados por el cuerpo docente de la facultad son de excelencia, en gran medida gracias a Joos Heintz, Verónica Becher, Santiago Figueira, Flavia Bonomo, Santiago Ceria, Diego Fernández Slezak, Javier Marengo, Min Chih Lin, Diego Garbervetsky, Pablo Turjanski, Sebastián Uchitel, Marina Groshaus, Isabel Méndez Díaz, Paula Zabala, Irene Loiseau, Esteban Feuerstein, Rodolfo Baader, Victor Braberman, Nicolás D'Ippolito, Alejandro Furfaro, Agustín Gravano, Juan Pablo Galeotti, Carlos Lopez Pombo, Guido de Caso, Hernán Melgratti, Esteban Mosckos, Fernando Schapachnik, Francisco Soullignac, Martín Urtasun y Hernán Wilkinson.

Gracias a los geniales compañeros de cursada que compartieron conmigo casi 9 años de carrera académica. Los asados con Ivan Postolski, Leandro Lera Romero,

March Martín Miguel, Matías Incem, Federico Lebrón, Pablo Gauna, Nicolás Maffongelli, Ariel Camblor y Juan Caamaño. Las geniales charlas de temas globales y profundos con Michel Mizrahi. Los mates y charlas de campo con Mache Bianchetti. La didáctica con Herman Schinca. La búsqueda constante de saber más con Gastón Bengolea. La seguridad con Joaco Rinaudo y Aye Chavez. El humor de Julián Sackmann y Mati Bender. El deporte con Dani Foguelman. El biohacking con Manu Giménez. Las Chicas en Tecnología de Caro Hadad, quien es una inspiración para todos. Las optimizaciones de Diego Delle Donne, Javi Martínez Viademonte, Agustín Montero, Brian Curcio y Fede Pousa. La genialidad matemática de Xavi Warnes. Los bits y buses de David González Márquez. Las neurociencias con Facu Carrillo y la lógica con Facu Carreiro. El espacio con Kevin Allekotte. Los temas de la vida con Diego de Estrada, quién ha despertado mi interés por Exactas. Las traducciones y biología con Ceci Herbert. Las intensas discusiones de Álgebra 1 con Marco Vanotti. A las hermanas Flor y Michelle Grenier, Andrés Furlan, Fran Leto Mera, Nico Rosner, Fede Carrá, Vivi Cotik y muchos otros que seguro me estoy olvidando.

A la gente del Laboratorio de Sistemas Complejos: Nahuel Olaiz, Cecilia Suárez, Guillermo Marshall, Diego Fernández Slezak, Esteban Mosckos, Damian Montaldo y David González Márquez, con quienes dí mis primeros pasos académicos.

A la gente del Instituto de Cálculo, con quien compartí oficinas y pasillos durante mis estudios. A Guillermo Durán, Vero Moyano, Flor Fernández Slezak, Juan Domingo, Agustín Álvarez, Alejo Salles, Lucas Bali, Alejandra Martínez, Leonardo Boechi, Andrés Muñoz (quien fue docente mío en el CBC), Francisco Wesner, Mariela Sued, Daniela Rodríguez, Víctor Yohai y otros que me estoy olvidando.

A mis entrenadores, Tigran Karhanyan y Vladimir Schevtsov, que han hecho muchísimo para que sea la persona que soy.

A mis amigos de siempre, Doug, Ale, Mati, Cris, Juan, Nico y Andy.

A Dani, quien siempre me apoyó en todo. Gracias.

A Vivi, quien ha contribuido a esta tesis no solo con su paciencia, sino con una buena parte de los gráficos de este documento.

Por último, quiero dar las gracias a mis padres, quienes me han dado el apoyo necesario y más que suficiente para poder dedicar este tiempo a formarme. Sin ustedes esto no hubiese sido posible. ¡Gracias!

Contents

I	Resúmenes en Español	iii
	Resumen	iv
	Introducción	v
	Transversal de cliques	viii
	Conjunto clique independiente	ix
	Recubrimiento por vecindades cerradas	x
	Conjunto vecindad independiente	xi
	Matching inducido	xii
	Conclusiones	xiii
II	Thesis	xv
	Abstract	xvi
	Agradecimientos	xvii
1	Introduction	2
1.1	Definitions and notations	3
1.1.1	Treewidth	6
1.2	Approximation algorithms	8
1.2.1	Greedy algorithms	11
1.2.2	Local ratio	12
1.2.3	Baker's technique	15
1.3	The problems	17
1.3.1	Clique transversal set	17

1.3.2	Clique independent set	19
1.3.3	Neighborhood covering set	20
1.3.4	Neighborhood independent set	21
1.3.5	Induced matching	22
1.4	Overview of our results	24
2	Induced Matching	26
2.1	A Δ -approximation algorithm	26
2.2	A greedy algorithm	31
2.2.1	$K_{1,k}$ -free graphs	33
2.2.2	Subclasses of chair-free graphs	33
2.3	Experimental results	35
3	Clique Transversal Set	37
3.1	Line graphs	37
3.1.1	Unweighted case	38
3.1.2	Weighted case	40
3.2	A simple 3-approximation for planar graphs	40
3.3	Weighted bounded treewidth graphs	42
3.4	An approximation scheme for planar graphs	43
3.5	Bounded degree graphs	45
4	Clique Independent Set	48
4.1	Line graphs	48
4.2	Weighted bounded treewidth graphs	51
4.3	An approximation scheme for planar graphs	55
5	Neighborhood Covering Set	58
5.1	Weighted bounded treewidth graphs	58
5.2	An approximation scheme for planar graphs	67
5.3	Further remarks	70
6	Neighborhood Independent Set	72
6.1	Inapproximability	72
6.2	A Δ -approximation algorithm	73
6.3	An approximation algorithm for diamond-free graphs	75
6.4	A polynomial time algorithm on cographs	77

6.5	A bound for regular graphs	79
6.6	Weighted bounded treewidth graphs	80
6.7	An approximation scheme for planar graphs	90
7	Conclusions and further work	92
7.1	Results	93
7.2	Further work	94

Nomenclature

$\alpha_C(G)$ The clique independence number of G .

$\alpha_N(G)$ The neighborhood independence number of G .

$\delta(e)$ If e is an edge uv , then $\delta(e) = \delta(u) \cup \delta(v)$.

$\Delta(G)$ The maximum degree of a vertex in G . If it is unambiguous, we simply write Δ .

$\delta(v)$ The edges incident to v , $\{uv : uv \in E\}$.

$\mathcal{K}(G)$ The set of all maximal cliques of G .

$\overline{G}$ The complement graph of G .

$\rho_N(G)$ The neighborhood covering number of G .

$\tau_C(G)$ The clique transversal number of G .

$\text{tw}(G)$ The treewidth number of G .

$C(e)$ The conflicting elements with element e .

C_n The cycle graph on n vertices.

$G - W$ The graph resulting of removing the vertices of W from G .

$G \cup H$ The union graph of graphs G and H .

$G \oplus H$ The join graph of G and H . It is the graph resulting of connecting each vertex of G with each of H .

$G[v] = (V[v], E[v])$ The graph induced by the closed neighborhood of v , $G[N[v]]$.

$G[W]$ The subgraph of G induced by W . It is the graph resulting of removing all vertices of $V \setminus W$ from G .

G_B If B is a node in a rooted tree decomposition T of G , then G_B is the subgraph of G induced by the vertices of the nodes in T^B .

- K_n The complete graph on n vertices.
- $K_{i,j}$ The complete bipartite graph with i vertices on the one side and j on the other.
- $L(G)$ The line graph of G .
- L_i The vertices of the i -th layer of a plane graph G .
- m The number of edges of a graph G , if clarification is needed, we write m_G .
- n The number of vertices of a graph G , if clarification is needed, we write n_G .
- $N(e)$ If e is an edge uv , then $N(e) = N(u) \cup N(v)$.
- $N(v)$ The open neighborhood of v .
- $N[v]$ The closed neighborhood of v , that is $N[v] = N(v) \cup \{v\}$.
- T^B If T is a rooted tree decomposition of G , then T^B is the subtree of T rooted at B .
- $x(A)$ If x is a vector defined over $B \supseteq A$, then $x(A) = \sum_{a \in A} x_a$.

Chapter 1

Introduction

*“The man who grasps principles
can successfully select his own
methods. The man who tries
methods, ignoring principles, is
sure to have trouble.”*

—Emerson

The roots of many optimization problems studied in computer science can be traced back to human activity. Unlike many arising in natural sciences as physics or biology, most of these problems are of discrete nature: we are to find an optimal solution within a finite set of feasible solutions. Therefore, a significant number of mathematical tools for modeling and optimizing problems over the real or complex fields are of little use for optimizing discrete problems. Developing useful tools and solving such problems is the main activity in the *discrete optimization* field.

Discrete optimization problems are ubiquitous. For example, finding the shortest route that a salesman should take for visiting all the shops in a city; assigning workers to jobs according to some criteria; scheduling a set of jobs on different machines to minimize the total completion time; monitoring every connection in a network with the smallest number of monitors; and many many other examples. Some of these problems are *easy* to solve using a computer, that is, there is an algorithm that computes an optimal solution in time bounded by a polynomial on the size of the input instance. Unfortunately, many of these problems are thought to be *hard*, in a sense that there is no polynomial time algorithm for solving them, unless some condition in complexity theory holds which many researchers consider unlikely. On a high level, this condition can be thought of as the equivalence between *finding* a feasible solution and *checking* whether some candidate solution is indeed feasible. We refer the reader to the classic Garey and Johnson book [41] for further presentation on this topic and from now on we assume the reader is familiar with the basic definitions involved in the P vs. NP question.

The fact that there are difficult problems is not an excuse for not dealing with them. Clearly, unless one is equipped with amazing insight, a superb mind and a great deal of luck, a reasonable step for tackling these problems is to sacrifice some desirable property. As it happens many times there are three desirable properties: efficiency, optimality and generality. Each choice is thoroughly researched. For example, sacrificing efficiency means giving no polynomial running time guarantees. Here we find the vast machinery for solving integer linear programs [25, 76, 80] and smartly designed exponential algorithms that run fast in practice or have small exponents [26, 37, 38, 89]. If we sacrifice generality, then we are restricting the instances of the problem we are going to solve. Such restrictions are useful for developing polynomial time algorithms [43]. Finally, if we sacrifice optimality we are in the realm of algorithms that output possibly suboptimal solutions. Here we can find randomized algorithms that work “well” with high probability [75]; the polynomial time heuristics with no quality guarantees of the output [42, 83]; and the heuristics that are provably good [86, 88]. In this thesis we focus on this last approach, that is, we give polynomial time algorithms and prove how good they perform compared to a theoretical optimum.

Many discrete optimization problems can be modeled with points (vertices) connected by lines (edges). These structures are called *graphs* and are surprisingly general. The most natural use of graphs is describing relations between elements. For example, we can model persons with vertices and add an edge between two individuals if they are friends in some social network—this models the social network itself. Usually, we want to find a set of vertices or edges with some property of interest. Most of times these sets may have a natural price we must pay for or get paid for, defining thus a discrete optimization problem.

The wide range of problems that can be modeled with graphs made research in graph theory a huge field in itself. Our results improve the understanding of these problems from an algorithmic and structural point of view.

In the following sections we formally introduce most of the concepts used in this thesis and review previous work.

1.1 Definitions and notations

Given a vector x over some finite set A we let x_a (or $x(a)$) be the value this vector takes on the element $a \in A$. Given a subset $B \subseteq A$, we define $x(B) = \sum_{b \in B} x_b$.

A *graph* is a pair $G = (V(G), E(G))$, where $E(G) \subseteq \{\{u, v\} : u, v \in V(G)\}$.

If the context is clear, we will use V instead of $V(G)$ and E instead of $E(G)$. The set V is the *vertex set* and E is the *edge set* of G . We let $n = |V|$ and $m = |E|$, when necessary we will write n_G and m_G . For convenience, we use the notation uv instead of $\{u, v\}$ for an edge $\{u, v\} \in E$. We say that u is *adjacent* to v whenever $uv \in E$. Given a vertex $v \in V$, we define the *open neighborhood* of v as $N(v) = \{u \in V : uv \in E\}$, and the *degree* of v as $\deg(v) = |N(v)|$. The *closed neighborhood* of v is the set $N[v] = N(v) \cup \{v\}$. Let $\Delta = \Delta(G) = \max\{\deg(v) : v \in V\}$ be the maximum degree of G . We say that an edge uv is *incident* to the vertices u and v . For each vertex $u \in V$, let $\delta(u) = \{uv : uv \in E\}$ be the set of edges with endpoint u . Given two subsets $V' \subseteq V$ and $E' \subseteq E \cap \{uv : u, v \in V'\}$ the graph (V', E') is a *subgraph* of G ; and if $E' = E \cap \{uv : u, v \in V'\}$, then (V', E') is an *induced* subgraph of G and we write $G[V'] = (V', E')$. Given a subset $W \subseteq V$, we let $G - W = G[V \setminus W]$ be the induced subgraph of G after removing the vertices in W . For every $v \in V$, let $G[v] = (V[v], E[v])$ be the subgraph induced by the closed neighborhood of v , that is $G[v] = G[N[v]]$.

Given two graphs, G and H , the *union graph* of G and H is defined as

$$G \cup H = (V(G) \cup V(H), E(G) \cup E(H)),$$

where the vertex and edge sets are disjoint. The *join graph* of G and H is defined as

$$G \oplus H = (V(G) \cup V(H), E(G) \cup E(H) \cup \{uv : u \in V(G), v \in V(H)\}).$$

A *complete graph* K_n is a graph with n vertices such that each pair of different vertices are connected. Note that we have $m = n(n-1)/2$. A *clique*¹ C in a graph G is a *maximal* subset of vertices that induces a complete subgraph with at least two vertices. For convenience, we also refer with the term “clique” to the induced subgraph. We let $\mathcal{K}(G)$ denote the set of all cliques of G .

A *cycle* is a graph $C_n = (V, E)$ with $n \geq 3$ and $V = \{v_1, \dots, v_n\}$ such that $E = \{v_1v_2, v_2v_3, \dots, v_{n-1}v_n, v_nv_1\}$. A *path* on n vertices, denoted P_n , results of removing one edge of a cycle on n vertices.

The *distance* between two vertices u and v is the number of edges in the shortest path between them. We let $\text{dist}(u, v)$ denote such value.

¹We remark that in some literature the term “clique” refers to *any* complete subgraph of G . In this work we use “clique” to denote only *maximal* complete subgraphs of size at least two.

A *connected component* C of G is a maximal subgraph of G such that there is a path between any two vertices in $V(C)$. A graph is *connected* if it has only one connected component.

A *tree* $T = (V, E)$ is a connected graph that contains no cycle as subgraph. A *rooted tree* T is a tree with a distinguished vertex or node r called *root*. If $uv \in E$ we say that u is the *parent* of v and v the *child* of u when $\text{dist}(r, u) < \text{dist}(r, v)$. A *leaf* is a vertex without children, and an *internal vertex* is a one that has at least one child.

Given a graph $G = (V, E)$, a subset $I \subseteq V$ such that no two vertices in I are adjacent is an *independent set* of G .

A *bipartite graph* is a graph whose set of vertices may be partitioned into two independent sets. Given two integers i and j , the *complete bipartite graph* $K_{i,j}$ is a bipartite graph with independent sets of size i and j , where each vertex on one side is connected to each vertex of the other. The complete bipartite graph $K_{1,3}$ is a *claw*.

Given a graph G we define the *complement graph* $\overline{G} = (V, \overline{E})$ where the edges missing in G are the edges of $\overline{G}$, i.e., $\overline{E} = \{uv : uv \notin E \text{ and } u \neq v\}$. If $\mathcal{G}$ is a graph class, we define $\text{co-}\mathcal{G}$ to be the class of graphs whose complement is in $\mathcal{G}$.

Given a graph family $\mathcal{F}$, we say that a graph G is $\mathcal{F}$ -free if G has no induced subgraph in the family $\mathcal{F}$. Note that if $\mathcal{F}$ is a finite family, then we can check in polynomial time whether G is $\mathcal{F}$ -free: for each graph $F \in \mathcal{F}$ we can take all the induced subgraphs of G with n_F vertices, which are exactly $\binom{n_G}{n_F} \in O(n_G^{n_F})$. Since F is fixed, checking if the subgraph is isomorphic to F is $O(1)$. Applying this procedure for each $F \in \mathcal{F}$ implies a polynomial time algorithm for the recognition problem of $\mathcal{F}$ -free graphs.

The *line graph* of G is a graph $L = L(G)$ whose vertex set is $E(G)$ and two vertices in L are adjacent if and only if their corresponding edges in G share a vertex. Formally, $E(L) = \{ef : \exists v \in V(G) \text{ such that } \{e, f\} \subseteq \delta_G(v)\}$.

A *planar graph* G is a graph that can be drawn on a plane such that no two edges cross each other. We call such a drawing an *embedding* and we refer to it as a *plane graph*. A plane graph for which all vertices lie on the outer region (or face) is *outerplanar*. A planar graph is k -outerplanar if the graph resulting of removing all the vertices on the outer face is $(k - 1)$ -outerplanar. We define the 1-outerplanar graphs as the outerplanar graphs.

A *cograph* is a graph that can be constructed using the following rules.

- A graph with a single vertex is a cograph.

- If G and H are cographs with disjoint vertex set, then $G \cup H$ is a cograph.
- If G is a cograph, then $\overline{G}$ is a cograph.

Another recursive definition is the following.

- $(\{v\}, \emptyset)$ is a cograph.
- If G and H are cographs, then $G \oplus H$ is a cograph.
- If G and H are cographs, then $G \cup H$ is a cograph.

A tree representing the above decomposition is called *cotree*, and it can be computed in linear time due to an algorithm by Corneil et al. [28]. Alternatively, a graph G is a cograph if and only if G is P_4 -free [27].

The remaining definitions and notation used in this work will be given when needed.

1.1.1 Treewidth

In this subsection we introduce and discuss a measure of tree-likeness of graphs known as *treewidth*. We first give a short motivation for these definitions, for a more in depth reading we suggest [10–13, 61]

Many optimization problems can be solved on trees using a dynamic programming approach. This follows from the recursive nature of rooted trees: either a vertex is a leaf, or it has a set of subtrees pending from it. If we can efficiently optimize some problem on the leaves of the tree and we know how to efficiently compute an optimal solution for an internal vertex based on the optimal solutions of its children, then by induction we can solve efficiently the problem for the whole tree.

If the input graph is not a tree, then the above dynamic programming ceases to be useful because we lose the recursive structure. However, there are many graphs that are very “similar” to trees. Similar in the sense that there is a way of representing them such that they look like trees if we “zoom-out enough.” By this we mean that there is some other tree which enables us to embed the original graph into this tree such that the more “tree-like” the original graph, the “neater” the embedding. This “neatness” tells us how much we must zoom-out to “confuse” the original graph with a tree. Such an embedding is a decomposition of the graph into several subsets of vertices connected in a tree structure that must have some

If the treewidth of a graph is bounded by a fixed k , then obtaining a tree decomposition of width at most k can be done in linear time by using the algorithm introduced in 1993 by Bodlaender [9].

A very celebrated and profound result by Bruno Courcelle states that if some graph has treewidth at most k , and the problem we are trying to optimize can be expressed in a certain logic², then there is a linear time algorithm for optimizing the cardinality version of this problem [29].

However, when dealing with weighted generalizations of some problems the Courcelle's result does not seem to work. Thus, solving the problems using dynamic programming explicitly becomes the most reasonable approach. For this purpose, a more comfortable tree decomposition might be used than the presented above.

A *nice* tree decomposition is a rooted tree decomposition that has only the following four possible node types.

- *Leaf*: a node B with only one element and no children.
- *Join*: a node B with two children B_1 and B_2 , such that $B = B_1 = B_2$.
- *Introduce*: a node B with exactly one child B' , such that $B = B' \cup \{v\}$ and $v \notin B'$.
- *Forget*: a node B with exactly one child B' , such that $B' = B \cup \{v\}$ and $v \notin B$.

An example is given in Figure 1.1.2.

If the treewidth of the graph is bounded by a fixed constant k , then given a tree decomposition one can obtain in linear time a nice tree decomposition with the same width and with $O(n)$ nodes [58].

We introduce some notations repeatedly used in the following chapters regarding tree decompositions. For a node B in the decomposition let T^B be the subtree of T rooted at B . Let $G_B = (V_B, E_B)$ be the subgraph of G induced by $V_B = \bigcup_{B' \in T^B} B'$, where $B' \in T^B$ means that B' is a node in the tree T^B .

1.2 Approximation algorithms

In this section we introduce the basic definitions and discuss some notions of approximation algorithms. For a comprehensive reading on this topic we suggest the

²MSOL: Monadic Second Order Logic. We omit the details of these results in this introduction.

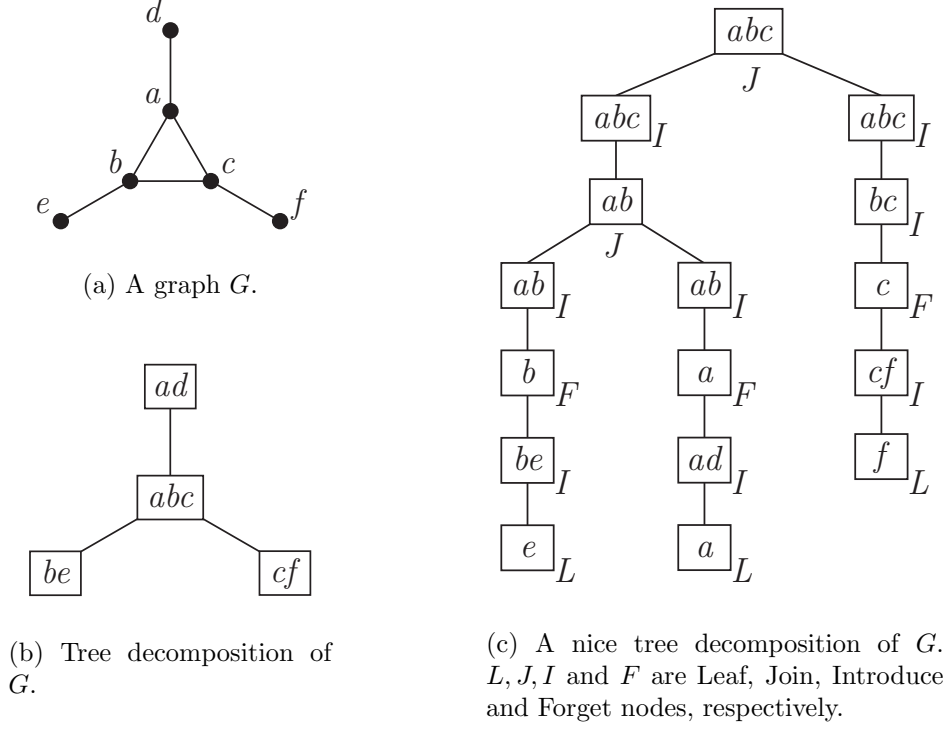


Figure 1.1.2: Nice tree decomposition.

book of Shmoys and Williamson [88].

Given a minimization problem Π , we say that an algorithm A is α -approximate or simply an α -approximation if for every instance I of Π we have $A(I) \leq \alpha OPT(I)$, where $OPT(I)$ is the optimal value of I and $A(I)$ is the value of the computed solution by A . In a similar way, if Π is a maximization problem, we say that A is α -approximate or an α -approximation if $A(I) \geq \frac{1}{\alpha} OPT(I)$ for each instance I . The number α is the *approximation ratio*, *performance ratio* or simply *factor*. In both cases we will have $\alpha \in [1, +\infty)$. The closer α is to 1, the better the approximation is. If $\alpha = 1$, then we have an exact algorithm. We say that Π admits a *polynomial time approximation scheme (PTAS)* if for any fixed constant $\epsilon > 0$ there is a $(1 \pm \epsilon)$ -approximation algorithm for Π . It is worth saying that in some works other conventions are used, for instance using $0 < \alpha \leq 1$ and asking that $A(I) \geq \alpha OPT(I)$ for maximization problems. In this thesis we use the former convention, except for approximation schemes for convenience.

The natural question is how can we prove that a computed solution is related by a multiplicative factor to the optimum. Since computing the optimum for the current instance is not an option because of time constraints, we are to use some

bound and relate it with the computed solution.

More formally, suppose that Π is a minimization problem and I an instance of Π . Suppose that $L(I)$ is a lower bound for $OPT(I)$ and we can efficiently compute a feasible solution of value $A(I) \leq \alpha L(I)$. Then,

$$OPT(I) \leq A(I) \leq \alpha L(I) \leq \alpha OPT(I).$$

Analogously, suppose that Π is a maximization problem and $U(I)$ is an upper bound for $OPT(I)$. If we can compute a solution of value $A(I)$ in polynomial time such that $U(I) \leq \alpha A(I)$, then

$$OPT(I) \geq A(I) \geq \frac{1}{\alpha} U(I) \geq \frac{1}{\alpha} OPT(I).$$

Note that in both cases $A(I)$ is an α -approximate solution for I . One observation to be made is that the bounds are not required to be computable in polynomial time—they may well be theoretical. The only requirement is the relation of the bound by a factor α with the computed solution. Therefore, the crucial step in designing approximation algorithms is obtaining bounds and algorithms which we can relate with these bounds.

Each of the above inequalities poses two important questions.

- How *tight* is α in the relation of the bound with $A(I)$?
- How *tight* is the relation of the bound with $OPT(I)$?

As expected, the tighter these inequalities, the better. In the first case, the tightness means that the analysis of the algorithm is *as good as possible* with respect to this bound. Since we look for a general analysis and not a per-instance approach, we say that *the analysis is tight* whenever there is at least one instance that attains equality. Moreover, in many cases we are satisfied with showing that there is an instance that *almost* attains equality. If we cannot exhibit such an instance, then we cannot guarantee that the analysis of the algorithm is *good* with respect to the chosen bound, meaning that there might be room for improving the analysis.

The second question is independent of the algorithm. In fact, it is an interesting question in itself, without any algorithm at hand. The maximum ratio over all instances between the optimum and the bound tells us the best possible approximation ratio we can hope for if we use this bound in the analysis. In other words, if we know that there is an instance such that the ratio between the optimal solution

and the bound is β , then we cannot design any approximation algorithm based on the given bound with ratio better than β . The maximum ratio between the optimal solution and the bound is commonly called *gap*. This idea has two practical applications.

The first scenario is when we already have an α -approximation for some problem and we are trying to find a new algorithm with a better approximation ratio. When a new bound is proposed, it is sensible to ask what the gap of this bound is. If the gap is bigger than α , then there is no reason in wasting time thinking algorithms that will use this bound because we will not be able to do any improvement with respect to α .

The second scenario occurs when we proposed an algorithm and proved that it is an α -approximation using the bound at hand. If we can see that the gap of the bound is close to α , then we know that we cannot possibly design *any* algorithm with an improved approximation ratio, unless we use a different bound with smaller gap.

In what follows we give the intuitions behind some techniques for designing approximation algorithms used in this thesis. We emphasize that in the following subsections there is some lack of mathematical rigor, this is so for the sake of clarity. Formal and rigorous proofs will be given when presenting the results.

1.2.1 Greedy algorithms

Greedy algorithms are among the most natural first attempts to solve many problems: iteratively make decisions that maximize the immediate benefit, disregarding what may come later. We refer the reader to [26, Chapter 16] for an introduction on this approach and to [88] for applications of this heuristic within the field of approximation algorithms.

In what follows we describe a common scenario without formalizing the details. Suppose we have a vertex-weighted graph $G = (V, E)$ with weights $w(v) \in \mathbb{Q}_{\geq 0}$ for every $v \in V$. Assume that our goal is to find a subset $U \subseteq V$ that maximizes the total weight $w(U) = \sum_{u \in U} w(u)$ such that U satisfies some desired property. One of the most natural approaches is to begin with an empty set U and iteratively add the heaviest possible vertex to U ensuring that we never violate the property. Eventually we reach a point where we cannot add any vertex to U , when this happens we return the computed set.

In many cases the computed solution U is feasible by design. Unfortunately,

for some problems such an algorithm can have an arbitrarily bad performance. However, in some cases we can prove that this approach is an approximation algorithm. Usually, a proof for the approximation ratio is based on a comparison of a hypothetical optimal solution with the computed one. The idea is that as we add some vertex to the set U , we “block” some previously unblocked candidate vertices from being eligible in further iterations. Which vertices are blocked depends on the property they must comply with. Let us denote $B(u)$ to the vertices blocked by u . It may be possible to show that the number of newly blocked vertices $|B(u)|$ is at most α . In fact, if S is some optimal solution, then it is enough to see that the number of blocked vertices that are as well in the hypothetical optimal S are at most α (i.e., $|B(u) \cap S| \leq \alpha$)—a weaker condition. This is usually proved by using some property that any feasible solution must have and the structure of the input. If this is true, it means that we err by at most α in each iteration. This follows from the fact that the weight of the added vertex is at least as big as the weight of each newly blocked vertex in the optimal solution, and we proved that these are at most α .

This same idea can be thought of as a sequence of bets whose benefit we immediately collect. The approximation factor is the ratio between the benefit $w(u)$ we obtain by making a bet on u and the price we pay for this bet (the weight of the blocked vertices for which we have not payed in previous bets). A more careful analysis is to pay for the bet no more than we could possibly win by betting otherwise, that is, we pay for the newly blocked vertices that are in $B(u) \cap S$, where S is some optimal solution.

1.2.2 Local ratio

Reuven Bar-Yehuda and Shimon Even championed the local ratio technique for approximation algorithms. We suggest the survey by Bar-Yehuda et al. on this topic [5].

We first introduce some vectorial notation to speak about problems and solutions. Let w be a vector (or function) representing the weights of the elements in some instance. Let x be a vector representing some solution to the instance. The vector x may be, for example, a vector in $\{0, 1\}^n$ taking the value $x_i = 1$ if the vertex v_i is in the solution and 0 otherwise. If x represents a feasible solution, then the value of the solution x is $w \cdot x = wx$. If the problem is formulated over graphs and the feasibility of x depends only on the graph structure and not upon

the weight vector w , then x will be feasible for any w ; however, for different weights w the optimum solution x may differ.

The local ratio technique is an approach based on the decomposition of the weight function into a sum of several functions or components. The idea is to find a feasible solution that is α -approximate with respect to each of these components. This will imply—under very general assumptions—that the solution is α -approximate with respect to the original function as well. For completeness, we reproduce the argument for the maximization version.

Theorem 1.2.1. (*Local Ratio, Bar-Yehuda et al. [5]*). *Let $w, w_1, w_2 \in \mathbb{R}^n$ such that $w = w_1 + w_2$. Let $x \in \mathbb{R}^n$ be a feasible α -approximate solution with respect to w_1 , and with respect to w_2 . Then, x is α -approximate with respect to w .*

Proof. Let x^*, x_1^* and x_2^* be optimal and feasible solutions with respect to w, w_1 and w_2 , respectively. By definition, $w_1 x_1^* \geq w_1 x^*$, and $w_2 x_2^* \geq w_2 x^*$. Therefore,

$$wx = w_1 x + w_2 x \geq \frac{1}{\alpha} w_1 x_1^* + \frac{1}{\alpha} w_2 x_2^* \geq \frac{1}{\alpha} w_1 x^* + \frac{1}{\alpha} w_2 x^* \geq \frac{1}{\alpha} wx^*.$$

■

In practice, the feasible solution is constructed recursively decomposing the weight vector into two vectors: the current (or local) and the recursive. By proving that the built solution is α -approximate with respect to the local vector and using an inductive reasoning for the recursive vector, one obtains an α -approximate solution with respect to the original vector.

It is interesting to note that the general greedy case discussed in [Subsection 1.2.1](#) can be seen as a particular case of local ratio. We use the same notation as in the previous subsection. We can think about this recursively: in each recursive step we choose a vertex u that maximizes $w(u)$ and make the weight decomposition given by $w = w_1 + w_2$, where for each $v \in V$

$$w_1(v) = \begin{cases} w(v) & \text{if } v \in B(u), \\ 0 & \text{otherwise,} \end{cases} \quad w_2(v) = \begin{cases} 0 & \text{if } v \in B(u), \\ w(v) & \text{otherwise.} \end{cases}$$

Let U be an α -approximation with respect to w_2 computed recursively. Note that $\{u\}$ is α -approximate with respect to w_1 because $|B(u) \cap S| \leq \alpha$ and $w_1(u) \geq w_1(z)$ for each $z \in B(u) \cap S$. It follows that $U \cup \{u\}$ is α -approximate with respect to w_1 , and with respect to w_2 . (We are assuming that $u \in B(v)$ if and only if $v \in B(u)$.)

Furthermore, we assume no vertex of weight zero is in the recursive solution U . Thus, no vertex in U can block u , and therefore $U \cup \{u\}$ is feasible.) It follows by Theorem 1.2.1 that $U \cup \{u\}$ is an α -approximation with respect to w .

However, the above vector notation and the Theorem 1.2.1 does not give much intuition by itself about how this approach works. Of course, we know that if we find a *suitable* decomposition, then we might get an approximation algorithm. One intuitive way of thinking about this is in terms of successive bets. For the above example the bet argument is straightforward: we pay at most $\alpha w(u)$ when we bet on u (that is an upper bound on what $B(u) \cap S$ costs), and we know that we will earn at least $w(u)$ because we pick u into the solution. The ratio, locally, is α , and hence the name of the technique. Note that each time we bet, we reduce the size of the problem with respect to w by the definition of w_2 . The bet itself is represented by w_1 .

The crucial observation is that in each bet the payment we make is some bound of an optimal solution restricted to $B(u)$ multiplied by some amount of weight. In the above case, the payment is $w(u)|B(u) \cap S| \leq \alpha w(u)$, where S is some optimal solution. If we could always find some u for which the bound involved is smaller than α , then we could improve the approximation ratio.

The problem for finding such u is that we do not have much information about S besides its optimality and feasibility. A way for circumventing this scarcity of information is to use some computable structure that will enable us to search *good* u 's and make a good bound. For this purpose, we can use the linear programming machinery. In particular, we can model the problem at hand as an integer program, relax the integrality constraints and use an optimal fractional solution x instead of a hypothetical S for crafting the bound. Assume, for the sake of the argument, that x can be computed in polynomial time. Note that

$$\sum_{v \in V} w(v)x_v = wx \geq w(S) = \sum_{v \in S} w(v).$$

If the linear program is “strong” enough, we may improve the bound by replacing $w(u)|B(u) \cap S|$ by

$$w(u) \sum_{v \in B(u)} x_v = w(u)x(B(u)).$$

Now that we have gained the possibility of exploring the information provided by x , suppose that we are able to show that there is always some u such that $x_u \neq 0$ and $x(B(u)) \leq \beta$. Let u be such a vertex. If we can guarantee that a bet on u will

cost us at most $w(u)x(B(u)) \leq \beta w(u)$ and we can always ensure that we earn at least $w(u)$ for this bet, then we have a β -approximation algorithm.

If we are to formulate this idea in a recursive way, we need to find a solution which is β -approximate with respect to some weight decomposition $w = w_1 + w_2$ representing the bet. Consider the following.

$$w_1(v) = \begin{cases} w(u) & \text{if } v \in B(u), \\ 0 & \text{otherwise,} \end{cases} \quad w_2(v) = \begin{cases} w(v) - w(u) & \text{if } v \in B(u), \\ w(v) & \text{otherwise.} \end{cases}$$

Note the subtle difference in the definition of w_1 : we assign to all the vertices in $B(u)$ the weight of u . As before, let U be a β -approximate solution with respect to w_2 . In this setting, we might have a problem that in the previous greedy approach we did not have: U may contain some vertex in $B(u)$, and therefore $U \cup \{u\}$ may not be feasible. If this is the case, then—if we look at U with weights w and not only w_2 —we gained at least $w(u)$ by means of some vertex in $U \cap B(u)$. If U does not contain any vertex of $B(u)$, then we can add u to U and thus earn $w(u)$ due to u . Therefore, either U or $U \cup \{u\}$ is a feasible β -approximation with respect to w_1 and w_2 , and therefore with respect to w by Theorem 1.2.1.

This idea of using a fractional solutions and locality is known as the *fractional local ratio* technique. The algorithms devised with this technique have little conceptual differences, since the key ingredient for obtaining good approximation ratios is the existence of vertices u such that $x_u \neq 0$ and $x(B(u))$ is *small*. Clearly, the bound on $x(B(u))$ depends upon the linear formulation.

As a final remark, when obtaining an approximation ratio for a fractional local ratio algorithm, the upper bound for proving the approximation ratio will be given by the value of a fractional solution to the linear program, i.e.,

$$\sum_{v \in V} w(v)x_v = wx.$$

This means that if the *integrality gap* of some linear formulation is larger than β for some instance, then we cannot expect a β -approximation with such formulation.

1.2.3 Baker's technique

In this subsection we introduce a technique for approximating many problems on planar graphs within arbitrarily good ratios. We first give some definitions. Then,

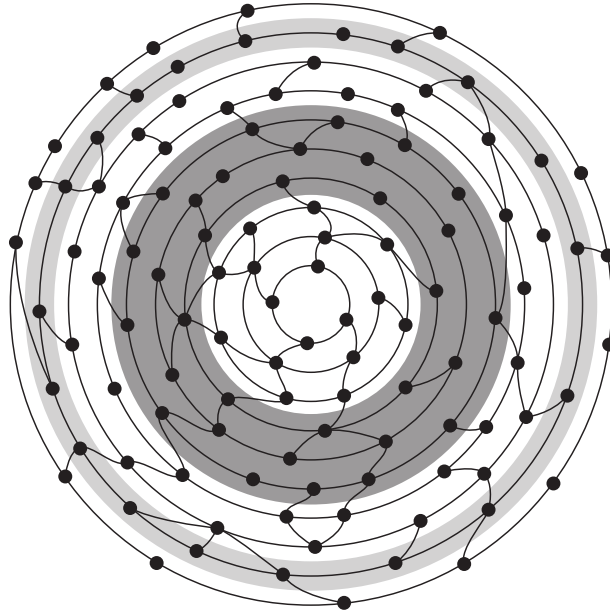


Figure 1.2.1: Layers and slices in a plane graph. The layer L_1 is shaded in light gray and the slice $G[L_4 \cup \dots \cup L_6]$ is shaded in a darker gray.

we give an intuitive explanation of how this method operates.

Let G be a plane graph. We define the *layers* L_i for $i \in \mathbb{Z}$ of G as follows: L_0 are the vertices that lie on the outer face of G , L_{i+1} are the vertices that lie on the outer face of $G - (L_0 \cup \dots \cup L_i)$, and $L_j = \emptyset$ for any $j < 0$ (we define this last case for notational convenience). We call *slice* to the subgraph induced by several consecutive layers, that is, $G[L_i \cup \dots \cup L_{i+t}]$. See Figure 1.2.1 for an example.

In 1994, Brenda Baker introduced a method for approximating optimization problems on planar graphs with performance ratio $1 \pm \epsilon$ for any fixed $\epsilon > 0$ [3]. This approach, which became known as the Baker’s technique, consists in separating G into several slices and solving the problem on these subgraphs, and afterwards extending these solutions to a global solution of G arguing that the incurred error is relatively small with respect to the number of layers in the slices. It turns out that many problems are polynomially solvable on slices with at most k layers, if k is a fixed constant. This is because slices with k layers are k -outerplanar graphs, and these in turn have bounded treewidth (see, e.g., the survey by Bodlaender [11]). Hence we can compute optimal solutions for each of the slices. The key observation is that a global optimal solution restricted to one slice may be at most as good as an optimal solution for the slice itself. Therefore, locally in each slice we outperform the optimal. However, we must pay a penalty for “gluing together” these sub-

solutions in order to obtain a feasible solution for the whole graph. Many times we can argue that the error incurred in this operation is *small* with respect to k , and thus the more layers the slices have, the better the approximation. There is, however, a significant drawback in this approach: the running time required for solving the problem on each slice usually grows exponentially on the number of layers in the slice. This poses the trade-off between speed and approximation ratio.

This trade-off calls for a natural question: what is the benefit in having such an algorithm which is “efficient” in theory, but which runs terribly bad for small ϵ ’s (i.e., big k ’s)? One optimistic answer is that it contributes to a theoretical characterization of this problem in terms of its approximation complexity. Some problems are hard to approximate within some ratio, and here we know that a ratio of $(1 \pm \epsilon)$ is attainable for any $\epsilon > 0$. The pessimistic answer is a bit more subtle: suppose we design an α -approximation algorithm for this problem on planar graphs which performs really well in practice, then the existence of the $(1 \pm \epsilon)$ -approximation algorithm tells us that we cannot hope to prove that there is no much room for improvement for α .

1.3 The problems

In this section we formally define the five problems we focus on in this thesis and overview the previous results.

1.3.1 Clique transversal set

Let $G = (V, E)$ be a graph. A subset $S \subseteq V$ such that $C \cap S \neq \emptyset$ for every clique $C \in \mathcal{K}(G)$ is a *clique transversal set* (CT-set). The *minimum CT-set* (CTS) problem is to find a CT-set of minimum cardinality. See [Figure 1.3.1](#) for examples. The *clique transversal number* $\tau_C(G)$ is the cardinality of such CT-set. The decision version of CTS is formulated as follows: given an integer k and a graph G , decide whether $\tau_C(G) \leq k$. The *weighted CT-set* (WCTS) problem is defined over vertex-weighted graphs, and the goal is to find a CT-set of G that minimizes its total weight.

A closely related parameter to $\tau_C(G)$ is the clique independence number $\alpha_C(G)$, which is the maximum number of disjoint cliques in G (see [Subsection 1.3.2](#) for a formal definition). These parameters are related by the following min-max duality relation: $\alpha_C(G) \leq \tau_C(G)$. If the equality holds for every induced subgraph of G ,

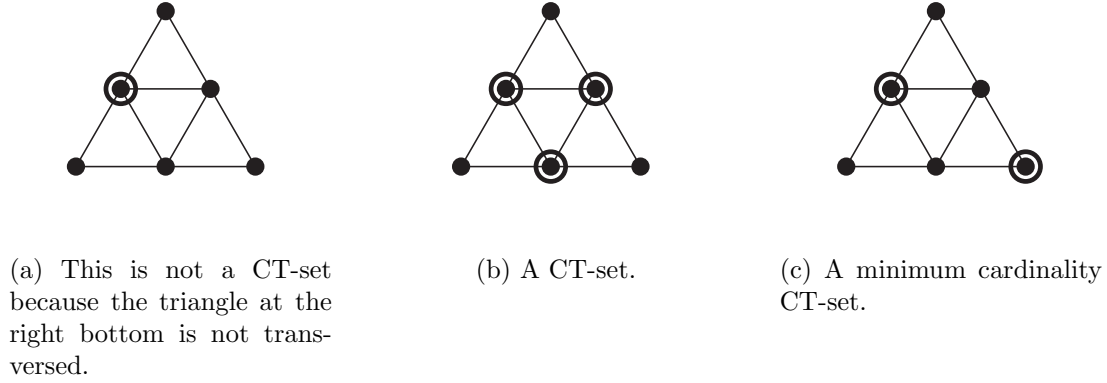


Figure 1.3.1: Clique transversal examples.

then G is *clique-perfect* [47].

The CTS problem was studied intensively since the publication [85] by Tuza in 1990. An extremal study of this problem was done in [36] by Erdős, Gallai and Tuza in 1991. The authors proved this problem to be NP-complete for triangle-free graphs and gave a linear time algorithm to find a clique transversal of size at most $n - \sqrt{2n} + \sqrt{2}$ for general graphs. To our best knowledge, it is unknown if this problem is in NP for general graphs because the number of cliques in a general graph can be exponential on $n + m$, and we do not know how to efficiently verify if a set of vertices is indeed a clique transversal, and thus for general graphs this problem is considered NP-hard. In 1993, Chang, Farber and Tuza proved this problem to be NP-complete for split graphs (where the number of cliques is polynomial) and gave a polynomial algorithm for strongly chordal graphs [21]. In the same year, Lee and Chang proved that distance-hereditary graphs are clique-perfect and as a by-product obtained a linear time algorithm for computing $\tau_C(G)$ for the same class [65]. In 1996, a polynomial algorithm for comparability graphs was given by Rangan et al. [4]. In 2000, Guruswami and Rangan proved that the problem remains NP-hard when restricted to co-comparability, planar and line graphs [47]. They were the first to attempt the weighted version of clique transversal—they obtained polynomial time algorithms for strongly chordal graphs, chordal graphs of bounded clique size and cographs. They also were the first to solve this problem for a non-clique-perfect class: they gave a $O(n^2)$ time algorithm for Helly circular-arc graphs. In 2002, Durán, Lin and Szwarcfiter proved that the problem of deciding whether a set of vertices is a clique transversal is co-NP-complete [35]. In 2008, Durán,

Lin, Mera and Szwarcfiter gave a general algorithm for computing a minimum-cardinality clique transversal which runs in $O(n^{2\tau_C(G)})$ time [34]. This algorithm is used to compute a minimum-cardinality clique transversal in $O(n^4)$ time for $\overline{3K_2}$ -free circular-arc graphs. They also developed two algorithms for Helly circular-arc graphs: for the weighted case their algorithm runs in $O(n^2)$ and for the unweighted case in linear time. In 2011, Liang and Shan proved that computing $\tau_C(G)$ and $\alpha_C(G)$ is NP-complete for cubic planar graphs of girth 3 [68]. For cubic graphs they gave a 1.96-approximation for computing $\tau_C(G)$. In 2015, the same authors presented a polynomial time algorithm for computing $\tau_C(G)$ for claw-free graphs with degree at most 4 [69]. In 2017, Liang et al. gave a polynomial time algorithm for (claw, K_4)-free planar graphs [70].

In 2001, Chang, Kloks and Lee gave a PTAS for transversing all the cliques of *maximum* size on an unweighted planar graph with the minimum number of vertices [23]. 10 years later, Lee gave a linear time algorithm for transversing all the cliques of *maximum* size for weighted graphs with bounded treewidth [64].

It is worth noting that with the reduction given by Bertossi [8] the dominating set problem on split graphs is at least as difficult to approximate as the set cover problem. On the other hand, it is easy to see that the size of a minimum-cardinality dominating set equals $\tau_C(G)$ for each split graph G . Thus, CTS is at least as difficult to approximate as the set cover problem.

1.3.2 Clique independent set

Let $G = (V, E)$ be a graph. A subset $\mathcal{S} \subseteq \mathcal{K}(G)$ such that $C \cap C' = \emptyset$ for any two different cliques C and C' in $\mathcal{S}$ is *clique independent* (CI). The *maximum CI-set* (CIS) problem is to find a maximum cardinality CI-set. See Figure 1.3.2 for examples. The *clique independence number* $\alpha_C(G)$ is the size of such CI-set. The decision version of CIS is formulated as follows: given an integer k and a graph G , decide whether $\alpha_C(G) \geq k$. The *weighted CI-set* (WCIS) problem is defined over vertex-weighted graphs. Suppose the weights of the vertices are w_v for $v \in V$. For each clique $C \in \mathcal{K}(G)$ we let $w_C = \sum_{v \in C} w_v$ be the weight of C . The goal in the WCIS problem is to find a CI-set $\mathcal{S}$ that maximizes $\sum_{C \in \mathcal{S}} w_C$.

The CIS problem is NP-complete when restricted to line graphs with maximum degree at most 3 [47], and when restricted to cubic planar graphs of girth 3 [68].

Observe that $\alpha_C(G) \leq \tau_C(G)$ for any graph G .

To our best knowledge, there are no previous approximation results for this

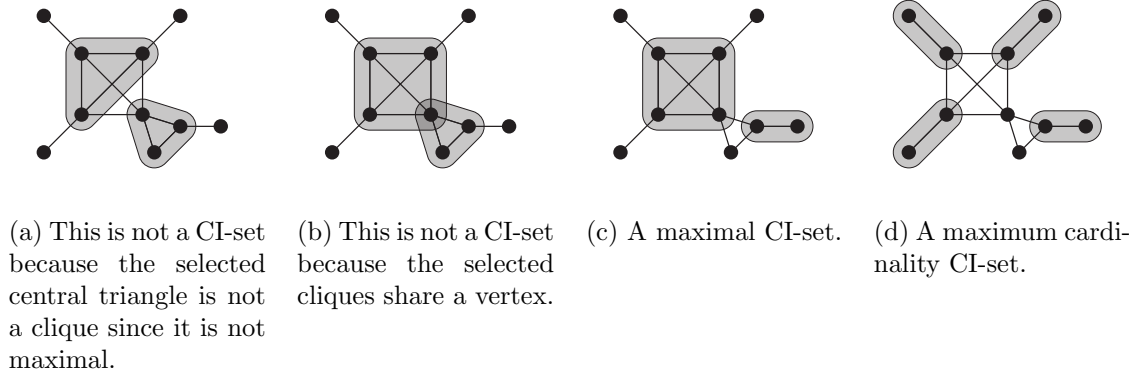


Figure 1.3.2: Clique independent set examples.

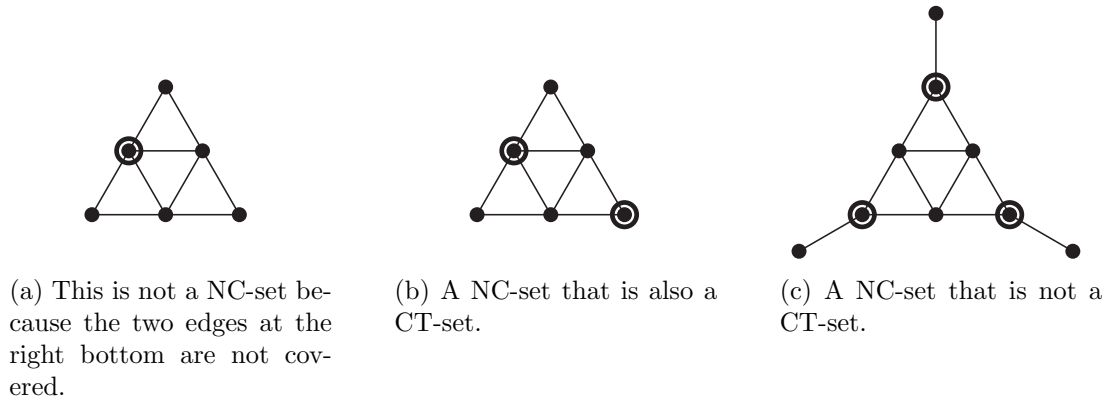


Figure 1.3.3: Neighborhood covering set examples.

problem.

1.3.3 Neighborhood covering set

Let $G = (V, E)$ be a graph. A subset $S \subseteq V$ such that every edge $e \in E$ is in $E[v]$ for some $v \in S$ is *neighborhood covering (NC)*. The *minimum NC-set (NCS)* problem is to find a NC-set of minimum cardinality. See [Figure 1.3.3](#) for examples. The *neighborhood covering number* $\rho_N(G)$ of G is the minimum cardinality of a NC-set. The decision version of NCS is formulated as follows: given an integer k and a graph G , decide whether $\rho_N(G) \leq k$. The *weighted NC-set (WNCS)* problem is defined over vertex-weighted graphs and the goal is to find a NC-set that minimizes its total weight.

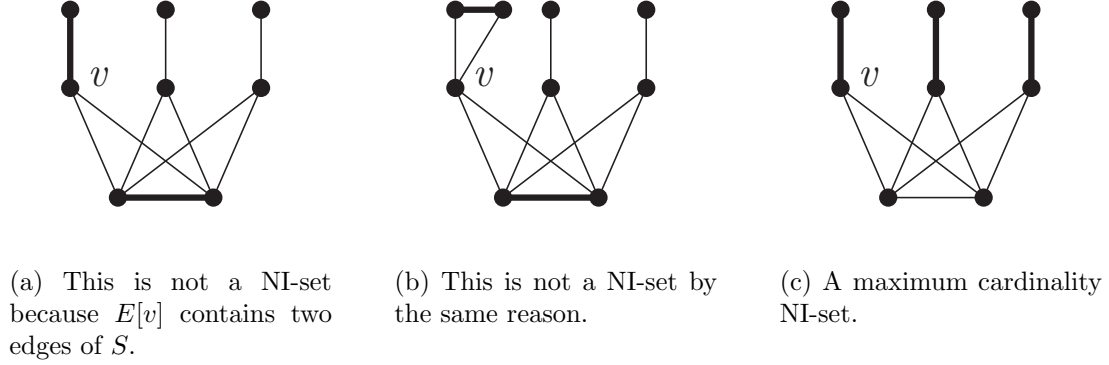


Figure 1.3.4: Neighborhood independent set examples. S is represented by bold edges.

The NCS problem is NP-complete when restricted to planar graphs [47]. To our best knowledge, there are no previous approximation results for this problem.

1.3.4 Neighborhood independent set

Let $G = (V, E)$ be a graph. A subset $S \subseteq E$ is *neighborhood independent (NI)* if $|E[v] \cap S| \leq 1$ for any vertex $v \in V$. See Figure 1.3.4 for an example. The *maximum NI-set (NIS)* problem is to find a NI-set of maximum cardinality. The *neighborhood independence number* $\alpha_N(G)$ of G is the size of such NI-set. The decision version of the problem is formulated as follows: given an integer k and a graph G , decide whether G contains a NI-set of size at least k (i.e., decide whether $\alpha_N(G) \geq k$). The *weighted NI-set (WNIS)* problem is defined over edge-weighted graphs and the goal is to find a NI-set that maximizes its total weight.

To our best knowledge, the WNIS problem was not studied before. The unweighted version was explored in the literature: In 1986, Lehel and Tuza [67] gave a linear time algorithm for on interval graphs. Wu [90] gave a $O(n^3)$ algorithm for strongly chordal graphs. Tuza et al. [21] proved the problem to be NP-complete on split graphs whose vertices of the independent set have degree 3; and gave a linear time algorithm for strongly chordal graphs if a strong elimination order is given as input. Guruswami and Rangan [47] proved the problem to be NP-complete for diamond-free planar graphs with $\Delta = 3$. In the same work it is shown that the problem is NP-complete on line graphs with $\Delta = 3$. Warnes [33, 87] gave a linear time algorithm for tree-cographs and P_4 -tidy graphs, and proved the problem to be

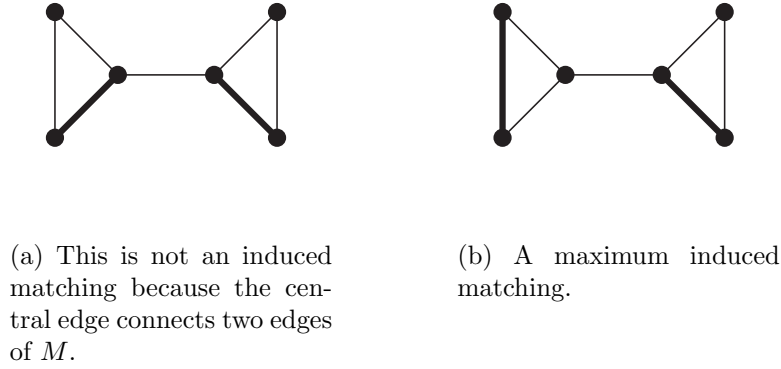


Figure 1.3.5: Induced matching examples. The set M is represented by bold edges.

NP-complete on co-bipartite graphs. Other non-algorithmic results related to this problem can be found in [48, 66]. A natural generalization of the NI-set problem was considered in [53]. To our best knowledge, no approximation algorithms for NIS or WNIS were explored before.

Observe that $\alpha_N(G) \leq \rho_N(G)$ for any graph G .

1.3.5 Induced matching

Let $G = (V, E)$ be a graph. A subset $M \subseteq E$ is an *induced matching* if M is a matching and no edge of $E \setminus M$ connects two endpoints of edges in M . See Figure 1.3.5 for examples. The *maximum induced matching (MIM)* problem is to find an induced matching of maximum cardinality. The *maximum weight induced matching (MWIM)* is to find an induced matching of an edge-weighted graph that maximizes its total weight.

In 1982, Stockmeyer and Vazirani studied some generalizations of the classical maximum matching problem, among which was MIM, although they referred to this problem as “risk-free” marriage problem. They proved it to be NP-complete when restricted to bipartite graphs of degree at most 4 [81]. Independently, this problem was introduced by Cameron in 1989, who gave a polynomial time algorithm for chordal graphs [16]. Thereafter MIM was polynomially solved on trees [39], circular-arc graphs [44], trapezoid, k -interval dimension, cocomparability graphs [45], $(\text{Star}_{1,2,3}, \text{Sun}_4)$ -free bipartite graphs [72], weakly chordal graphs [18] and on bounded treewidth graphs [74]. Cameron exploited the relations between the

families of G and $L(G)^2$ to conclude that MIM is polynomially solvable for polygon-circle, AT-free, and filament-interval graphs; where the latter contains cocomparability, circle, circular-arc, chordal and outerplanar graphs [17]. A polynomial time algorithm for hhd-free graphs and a linear time algorithm for a subclass of hhd-free graphs which is a more general class than chordal graphs were given in [62]. The problem was proven to be NP-complete on bipartite graphs of maximum degree 3, C_4 -free bipartite graphs [72], d -regular graphs for $d \geq 5$, line graphs (which implies that MIM is NP-complete on claw-free graphs and chair-free graphs) [60] and on cubic planar graphs [32, 59]. Chang gave several algorithms for classes related to AT-free graphs, in particular he proved the MWIM problem to be polynomially solvable on graphs with bounded asteroidal index [22]. Takaoka et al. observed that orthogonal ray graphs have bounded asteroidal index [82], which by [22] implies a polynomial time algorithm for MWIM for this class. In 2003, Kobler and Rotics solved MIM in polynomial time for line graphs of Hamiltonian graphs, and proved that the problem remains NP-complete on Hamiltonian graphs [60]. In the same paper they gave some polynomial time algorithms for subclasses of P_5 -free graphs and they noted that if G is P_5 -free, then $L(G)^2$ is P_5 -free. In 2003 it was still an open question whether the maximum independent set on P_5 -free graphs was polynomial time solvable, and therefore this observation did not directly yield a polynomial time algorithm for MIM on P_5 -free graphs. Remarkably, this long standing question was affirmatively answered by Lokshantov et al. in 2014 [71]; which implies that MIM is polynomial on P_5 -free graphs. In [60] a polynomial time algorithm was given for recognizing graphs where the size of a maximum induced matching is the same as the size of a maximum matching, and a polynomial time algorithm was given to find a MIM in such class. A simpler recognition algorithm for this class was given in [19]. The line of research of [19, 60] was continued by Duarte et al. in [31], where they simplified further the proofs of [19] and gave a polynomial time algorithm for recognizing graphs where the maximum induced matching differs in at most k with the maximum matching. In 2011, Brandstädt and Mosca explored some generalizations of MIM and proved that MWIM is polynomially solvable on chordal graphs, (claw, net)-free graphs and some other subclasses of claw-free graphs [15].

Regarding approximability, for graphs of degree at most Δ there is only a $2(\Delta - 1)$ -approximation algorithm given by Zito in 1999 [91]. However, several approximations were given for d -regular graphs with $d \geq 3$, which all used the same theoretical upper bound for proving their approximation ratios—namely, that any induced matching of a d -regular graph has at most $m/(2d - 1)$ edges. This

bound was first introduced by Zito in [91], where he gave a simple greedy algorithm with performance ratio $d - \frac{1}{2} + \frac{1}{4d-2}$. This was improved by Duckworth et al. in 2005 [32], where the authors gave an asymptotic approximation ratio $d - 1$. In the same work they gave a PTAS for planar graphs of degree at most 3. In the following year, Gotthilf and Lewenstein gave a $0.75d + 0.15$ approximation ratio for d -regular graphs [46]. In 2015, this ratio was further improved to $0.708\bar{3}d + 0.425$ on (C_3, C_5) -free d -regular graphs by Rautenbach [78]. In 2014, Joos et al. gave an algorithm for cubic graphs with performance ratio $9/5$ [56].

Several results were given regarding lower bounds and algorithms attaining them. Kang et al. proved that any subcubic planar graph has an induced matching of size at least $m/9$ and gave an algorithm that finds such a solution [57]. Henning and Rautenbach proved that in subcubic graphs without short cycles there must be an induced matching of size at least $(n - 1)/5$ [51]. Joos considered the bounded degree graphs and gave a polynomial time algorithm that computes an induced matching of size at least $\frac{n}{(\lceil \Delta/2 \rceil + 1)(\lfloor \Delta/2 \rfloor + 1)}$ for graphs with sufficiently large Δ and with no isolated vertices [54]. Joos and Nguyen proved that one can find an induced matching in polynomial time with at least $m/20$ edges for graphs with degree at most 4, and at least $m/18$ edges for a subclass these graphs [55].

On the negative side, MIM cannot be approximated on general graphs with a constant performance ratio unless $P=NP$ [91]. Furthermore, the problem cannot be approximated within a factor $n^{1/2-\epsilon}$ for some $\epsilon > 0$, unless $P=NP$ [77]. Moreover, MIM is APX-complete for d -regular bipartite graphs for $d \geq 3$ [30].

Despite the vast amount of research done for induced matchings, not much has been achieved for the weighted version of this problem besides [15, 22, 82]. To our best knowledge, no approximation algorithm was given using a linear program as an upper bound. However, in 2007, Fukunaga and Nagamochi considered a generalization of MWIM with edge capacities, and for some particular cases (that excluded the classical induced matching) they gave some constant approximation ratios by relating a natural linear programming formulation with a capacitated b -matching polytope [40].

1.4 Overview of our results

In Chapter 2 we give a Δ -approximation algorithm for MWIM on general graphs. We furthermore explore the greedy approach and show results for $K_{1,k}$ -free graphs and subfamilies of this class.

In [Chapter 3](#) we explore the CTS and WCTS problems for different graph classes. In particular, for line graphs we give a 3-approximation algorithm for CTS and a 4-approximation algorithm for WCTS. Furthermore, we observe that there is a trivial 4-approximation for WCTS on planar graphs and we give a 3-approximation for CTS. To further improve this ratio we give a linear time algorithm for WCTS on bounded treewidth graphs. Using this algorithm we give a PTAS for WCTS planar graphs. Moreover, we show a $\lceil (\Delta(G) + 1)/2 \rceil$ -approximation for graphs whose maximum degree $\Delta(G)$ is bounded by a constant.

In [Chapter 4](#) we give a $\max \left\{ 3, \frac{\Delta(G)+3}{5} + \epsilon \right\}$ -approximation algorithm for CIS on general graphs. For the WCIS problem we give a linear time algorithm for bounded treewidth graphs. Using this result we give a PTAS for WCIS on planar graphs.

In [Chapter 5](#) we solve in linear time the WNCS problem on bounded treewidth graphs and use this for designing PTAS for planar graphs.

In [Chapter 6](#) we explore the WNIS problem giving a Δ -approximation algorithm for general graphs, a $(\Delta/2 + 1)$ -approximation for diamond-free graphs, and an exact polynomial time algorithm for cographs. Furthermore, we solve this problem on bounded treewidth graphs in linear time and give a PTAS for planar graphs.

Finally, in [Chapter 7](#) we give summarize our contributions and give possible lines for further research.

Chapter 2

Induced Matching

*“Fear those who crave power.
Join those who regard power as
a cross to be carried.”
—Nikita Mikhalkov*

Let $G = (V, E)$ be a simple undirected graph. A subset $\mathcal{M} \subseteq E$ is an *induced matching* if $\mathcal{M}$ is a matching and no edge of E connects two edges of $\mathcal{M}$. The *maximum induced matching (MIM)* problem is to find an induced matching of maximum cardinality. In this chapter we consider the weighted version of the problem: given an edge-weighted graph the *maximum weight induced matching (MWIM)* problem is to find an induced matching that maximizes its total weight.

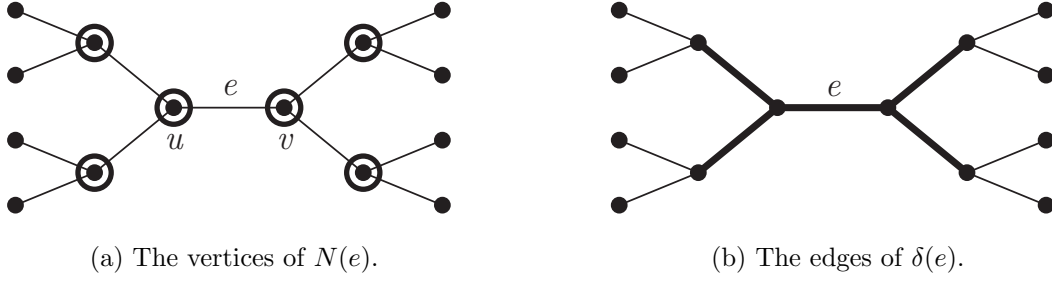
We propose a fractional local ratio algorithm for MWIM with performance ratio Δ . Furthermore, we show that a simple greedy algorithm yields an approximation ratio $2k - 3$ for $K_{1,k}$ -free graphs and k for $(K_{1,k}, \text{chair})$ -free graphs. Note that since any claw-free graph is $K_{1,k}$ -free for $k \geq 3$ and cannot contain a chair, it follows that MIM is NP-complete on $(K_{1,k}, \text{chair})$ -free graphs.

2.1 A Δ -approximation algorithm

Let $G = (V, E)$ be an edge-weighted graph with weights $w_e \in \mathbb{Q}_{\geq 0}$. For an edge $uv = e \in E$ we define $N(e) = N(v) \cup N(u)$ and we extend the notation of δ to edges as $\delta(e) = \delta(u) \cup \delta(v)$, where $N(v)$ is the open neighborhood of v and $\delta(v)$ is the set of edges incident to v . We let $C(e) \subseteq E$ be the set of edges which are *in conflict*¹ with e ; more formally,

$$C(e) = \bigcup_{w \in N(e)} \delta(w).$$

¹The intuition is that the edges of $C(e)$ cannot be in the same induced matching with e . However, since $e \in C(e)$, this would not be an accurate definition.

Figure 2.1.1: The conflict set $C(e)$.

Note that in our definition $e \in C(e)$. See [Figure 2.1.1](#) for an example. The intuition behind the following algorithm is given in [Subsection 1.2.2](#).

We can model the MWIM problem with an integer linear program. We define the binary variables x_e for $e \in E$ such that e is included in the solution if and only if $x_e = 1$. Consider the following formulation, where $x(A)$ denotes $\sum_{a \in A} x_a$.

$$\begin{aligned}
 \max \quad & \sum_{e \in E} w_e x_e \\
 \text{s.t.} \quad & x(\delta(e)) \leq 1 \quad \forall e \in E, \\
 & x_e \in \{0, 1\} \quad \forall e \in E.
 \end{aligned}$$

Our algorithm uses linear relaxations of the above program restricted to different sets of variables. Formally, for a subset $F \subseteq E$, we denote LP_F to the following linear program restricted to the variables x_f for $f \in F$.

$$\begin{aligned}
 \max \quad & \sum_{e \in E} w_e x_e \\
 \text{s.t.} \quad & x(\delta(e)) \leq 1 \quad \forall e \in E, \\
 & x_e \geq 0 \quad \forall e \in F.
 \end{aligned}$$

In what follows we show that [Algorithm 1](#) has performance ratio Δ provided that in each recursive call one can find an edge $e \in F$ such that $x(C(e)) \leq \Delta$. Note that there is at most one recursive call in each call, and in each step $|F|$ decreases by at least one. Therefore, the algorithm ends after at most $|E|$ recursive calls. For completeness, we reproduce the proof of the following theorem given in [\[20\]](#).

Theorem 2.1.1. ([\[5, 20\]](#)). *Let $F \subseteq E$ be a subset of edges. If for any non empty $F' \subseteq F$ and any feasible solution y to $LP_{F'}$ there is some $g \in F'$ such*

Algorithm 1 $\text{IM}(F, w)$

```

1: if  $F = \emptyset$  then
2:   return  $\emptyset$ 
3: Compute an optimal solution  $x$  to  $LP_F$ 
4: Let  $F_0 = \{e \in F : x_e = 0\}$ 
5: if  $F_0 \neq \emptyset$  then
6:   return  $\text{IM}(F \setminus F_0, w)$ 
7: Let  $e \in F$  such that  $x(C(e)) \leq \Delta$ 
8: For each  $f \in F$ , let  $\hat{w}_f = \begin{cases} w_e & \text{if } f \in C(e), \\ 0 & \text{otherwise.} \end{cases}$ 
9:  $\mathcal{M} \leftarrow \text{IM}(F \setminus \{e\}, w - \hat{w})$ 
10: if  $\mathcal{M} \cup \{e\}$  is an induced matching (i.e. if  $C(e) \cap \mathcal{M} = \emptyset$ ) then
11:    $\mathcal{M} \leftarrow \mathcal{M} \cup \{e\}$ 
12: return  $\mathcal{M}$ 

```

that $y(C(g)) \leq \Delta$, then Algorithm 1 computes an induced matching $\mathcal{M}$ such that $w(\mathcal{M}) \geq \frac{1}{\Delta} \sum_{f \in F} w_f x_f$, where x is the solution to LP_F computed in Line 3.

Proof. We prove this by induction in the number of iterations. The base case is handled in Line 2, which trivially holds.

Suppose the algorithm returns on Line 6. Let x' be the solution to $LP_{F \setminus F_0}$ computed in the recursive call. Then

$$w(\mathcal{M}) \geq \frac{1}{\Delta} \sum_{f \in F \setminus F_0} w_f x'_f \geq \frac{1}{\Delta} \sum_{f \in F \setminus F_0} w_f x_f = \frac{1}{\Delta} \sum_{f \in F} w_f x_f,$$

where the first inequality holds by inductive hypothesis; the second holds because x' is an optimal solution for $LP_{F \setminus F_0}$ and x restricted to $F \setminus F_0$ is feasible for $LP_{F \setminus F_0}$; and the last equality holds because $x_f = 0$ for each $f \in F_0$.

We now consider the case when the algorithm returns on Line 12. Let $\tilde{w} = w - \hat{w}$. Denote x' to the computed optimal solution to $LP_{F \setminus \{e\}}$ with weights $\tilde{w}$. On the one hand, we have

$$\tilde{w}(\mathcal{M}) \geq \frac{1}{\Delta} \sum_{f \in F \setminus \{e\}} \tilde{w}_f x'_f \geq \frac{1}{\Delta} \sum_{f \in F \setminus \{e\}} \tilde{w}_f x_f = \frac{1}{\Delta} \sum_{f \in F} \tilde{w}_f x_f,$$

where the first inequality follows from the inductive hypothesis; the second from the fact that x' is an optimal solution for $LP_{F \setminus \{e\}}$ (with weights $\tilde{w}$) and x restricted to $F \setminus \{e\}$ is feasible for $LP_{F \setminus \{e\}}$; and the last equality holds because $\tilde{w}_e = 0$, regardless of whether $e \in \mathcal{M}$ or not. On the other hand, we have

$$\hat{w}(\mathcal{M}) = \hat{w}_e |\mathcal{M} \cap C(e)| \geq \hat{w}_e \geq \hat{w}_e \frac{x(C(e))}{\Delta} = \frac{1}{\Delta} \sum_{f \in C(e)} \hat{w}_e x_f = \frac{1}{\Delta} \sum_{f \in F} \hat{w}_f x_f,$$

where the first equality follows from the definition of $\hat{w}$; the first inequality follows since $\mathcal{M} \cap C(e)$ is always non-empty; the second inequality because $x(C(e)) \leq \Delta$; and the last equality follows because $\hat{w}_f = 0$ for any $f \in F \setminus C(e)$ and $\hat{w}_f = \hat{w}_e$ for each $f \in C(e)$.

Therefore, we have

$$w(\mathcal{M}) = \tilde{w}(\mathcal{M}) + \hat{w}(\mathcal{M}) \geq \frac{1}{\Delta} \sum_{f \in F} \hat{w}_f x_f + \frac{1}{\Delta} \sum_{f \in F} \tilde{w}_f x_f = \frac{1}{\Delta} \sum_{f \in F} w_f x_f.$$

■

Lemma 2.1.1. *Let x be a solution to LP_F , then there is some edge $e \in F$ such that $x(C(e)) \leq \Delta$.*

Proof. We suggest Figure 2.1.2 for a clarification of the following argument.

Let $v \in V$ be a vertex that maximizes $\beta = x(\delta(v))$. Take any edge $uv = e \in \delta(v) \cap F$. For each $w \in N(v) \setminus \{u\}$, we have that

$$x(\delta(vw)) = x(\delta(v) \cup \delta(w)) \leq 1,$$

and therefore $x(\delta(w) \setminus \delta(v)) \leq 1 - x(\delta(v)) = 1 - \beta$. Hence,

$$\begin{aligned} x(C(e)) &\leq x(\delta(e)) + \sum_{w \in N(v) \setminus \{u\}} x(\delta(w) \setminus \delta(v)) + \sum_{w \in N(u) \setminus \{v\}} x(\delta(w)) \\ &\leq 1 + (\Delta - 1)(1 - \beta) + (\Delta - 1)\beta \\ &= \Delta, \end{aligned}$$

where the first inequality holds because each variable of $x(C(e))$ appears at least once in the sum; and the second inequality holds since $x(\delta(w)) \leq \beta$ for each $w \in N(u) \setminus \{v\}$ by definition of β . ■

Corollary 2.1.1. *Algorithm 1 has performance ratio Δ .*

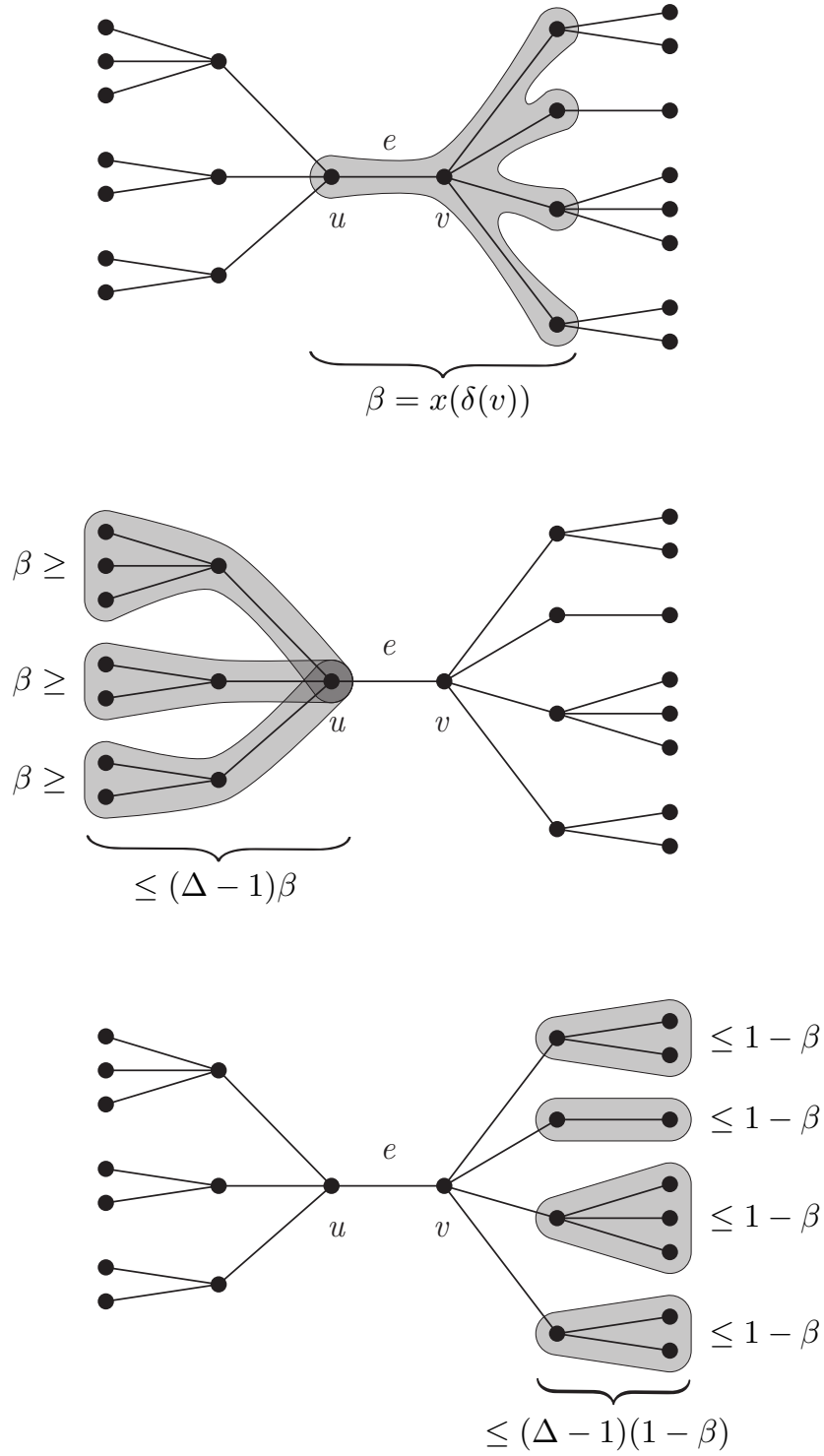


Figure 2.1.2: The bounding argument for Lemma 2.1.1.

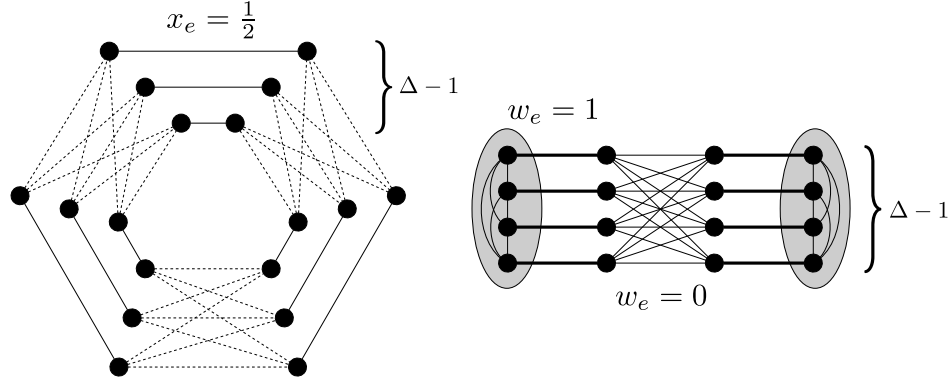


Figure 2.1.3: Tightness of the Algorithm 1 for graphs of degree at most Δ .

Remark 1. For any Δ there is a Δ -regular graph G and a subset of edges F such that the optimal solution x to LP_F is such that $x(C(e)) = \Delta - 1/2$ for each edge $e \in F$. In other words, Lemma 2.1.1 is almost tight.

Proof. Connect three matchings of $\Delta - 1$ edges by three complete bipartite subgraphs. Define the set F as the matchings. An example for $\Delta = 4$ is depicted in the first graph of Figure 2.1.3, where the solid edges are those in F . An optimal solution to LP_F is given by assigning $1/2$ to each edge in F and zero to those in $E \setminus F$, which implies $x(C(e)) = \Delta - 1/2$ for each $e \in F$. ■

Remark 2. The integrality gap is at least $\Delta - 1$. This means that unless we use a stronger linear formulation, we cannot hope to improve much the approximation ratio using the linear program as upper bound.

Proof. Consider two matchings of $\Delta - 1$ edges connected by a complete bipartite graph. With the remaining vertices of each matching form a clique. Assign a zero weight to the edges in the complete bipartite subgraph and the cliques, and a unitary weight to the edges of the matchings. An example for $\Delta = 5$ is depicted in the second graph of Figure 2.1.3, where the bold edges have weight 1. Clearly, an optimal induced matching has weight 1, while assigning $1/2$ to each edge of the matchings yields a fractional solution of weight $\Delta - 1$. ■

2.2 A greedy algorithm

Consider the natural greedy approach: sort the edges by their weights in a non-increasing order and iteratively keep adding the heaviest possible edge to the so-

lution, making sure that at every moment the solution is an induced matching. Return the constructed set once no more edges can be added.

Observe that this algorithm can be implemented in $O(m \log n)$ time: sorting is $O(m \log n)$; then we iterate through the edge set and we take the first edge uv whose endpoints are non-marked; we add this edge and we mark $N(u) \cup N(v)$ —this second phase takes $O(m)$ time.

Recall that $C(e) = \bigcup_{w \in N(e)} \delta(w)$. Note that if $\mathcal{M}$ is a maximum weight induced matching and $|\mathcal{M} \cap C(e)| \leq \alpha$ for each $e \in E$, then in each iteration of the greedy algorithm we pay at most α times what we receive for the edge we pick, and thus the approximation ratio is α . Formally, if $S = \{e_1, \dots, e_t\}$ is the induced matching constructed by the algorithm, where e_i was added before e_{i+1} , then

$$\begin{aligned} w(\mathcal{M}) &= w\left(\bigcup_{i=1}^t C(e_i) \cap \mathcal{M}\right) \\ &= w\left(\bigcup_{i=1}^t C(e_i) \cap \mathcal{M} \setminus \bigcup_{j=1}^{i-1} C(e_j)\right) \\ &= \sum_{i=1}^t w\left(C(e_i) \cap \mathcal{M} \setminus \bigcup_{j=1}^{i-1} C(e_j)\right) \\ &\leq \sum_{i=1}^t \alpha w_{e_i} \\ &= \alpha w(S), \end{aligned}$$

where the first equality holds because $\bigcup_{e \in S} C(e) = E$; the second is a rewriting of the edges of $\mathcal{M}$ in the order of their appearances in $C(e_1), \dots, C(e_t)$; the third equality holds because the above sets are disjoint; the inequality holds since $w_{e_i} \geq w_f$ for each

$$f \in \mathcal{M} \setminus \bigcup_{j=1}^{i-1} C(e_j) \subseteq E \setminus \bigcup_{j=1}^{i-1} C(e_j),$$

and since

$$|C(e_i) \cap \mathcal{M} \setminus \bigcup_{j=1}^{i-1} C(e_j)| \leq |C(e_i) \cap \mathcal{M}| \leq \alpha;$$

and the final equality is simply the definition of $w(S)$. We thus proved the following.

Theorem 2.2.1. *If for any induced matching $\mathcal{M}$ we have $|\mathcal{M} \cap C(e)| \leq \alpha$ for each $e \in E$, then the greedy algorithm has performance ratio α .*

2.2.1 $K_{1,k}$ -free graphs

Observe that if $k = 2$, then a $K_{1,k}$ -free graph is a union of cliques and isolated vertices. We thus consider the case $k \geq 3$.

Lemma 2.2.1. *If G is $K_{1,k}$ -free and $\mathcal{M}$ is an induced matching of G , then it holds that $|\mathcal{M} \cap C(e)| \leq 2k - 3$ for any edge $uv = e \in E$.*

Proof. If $e \in \mathcal{M}$, then $|C(e) \cap \mathcal{M}| = 1 \leq 2k - 3$. Suppose that $e \notin \mathcal{M}$. If $\delta(u) \cap \mathcal{M} \neq \emptyset$, then since $\mathcal{M}$ is an induced matching, it follows that $\delta(v) \cap \mathcal{M} = \emptyset$. Since G is $K_{1,k}$ -free, there are at most $k - 1$ pairwise nonadjacent neighbors of v , meaning that $|\delta(N(v)) \cap \mathcal{M}| \leq k - 1$, where $\delta(N(v)) = \bigcup_{w \in N(v)} \delta(w)$. Note that $\delta(N(u)) \cap \mathcal{M} \subseteq \delta(N(v))$, and therefore $|C(e) \cap \mathcal{M}| \leq k - 1 < 2k - 3$. Suppose now $\delta(u) \cap \mathcal{M} = \emptyset$. If $|C(e) \cap \mathcal{M}| \geq 2k - 2$, then it must be the case that $|\delta(N(u)) \cap \mathcal{M}| = |\delta(N(v)) \cap \mathcal{M}| = k - 1$ and these sets are disjoint, but this is not possible because in $N[u]$ there must be an induced $K_{1,k}$ centered on u with k pairwise disjoint vertices given by $k - 1$ endpoints of $\mathcal{M} \cap \delta(N(u))$ and v . ■

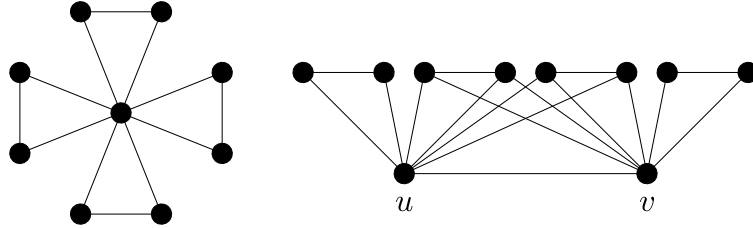
Corollary 2.2.1. *If G is $K_{1,k}$ -free ($k \geq 3$), then the greedy algorithm has performance ratio $2k - 3$.*

2.2.2 Subclasses of chair-free graphs

Suppose now that G is chair-free and $\mathcal{M}$ is an induced matching of G . In what follows we show that if there is some edge $e \in E$ such that $|\mathcal{M} \cap C(e)| = k \geq 4$, then G must have at least one of two possible graphs as induced subgraph.

Define G_k ($k \geq 3$) to be the graph given by an induced matching of k edges with a universal vertex. Let H_k ($k \geq 4$) be the graph given by an induced matching of size k and two additional adjacent vertices, u and v , such that u is connected to all the vertices of the first $k - 1$ edges in the matching, and v is connected to all the vertices of the last $k - 1$ edges in the matching (see [Figure 2.2.1](#) for an example). In what follows we say that an edge uv neighbors with a vertex w if $w \in N(v) \cup N(u) \setminus \{u, v\}$.

Lemma 2.2.2. *Let G be a chair-free graph and $\mathcal{M}$ be an induced matching of G . If $v \in V$ is not an endpoint of some edge in $\mathcal{M}$ and v neighbors with $k \geq 3$ edges from $\mathcal{M}$, then v and its neighboring edges in $\mathcal{M}$ induce a G_k .*

Figure 2.2.1: Graphs G_4 and H_4 .

Proof. Suppose for contradiction that there is some edge $uw \in \mathcal{M}$ such that $uv \in E$ but v and w are not adjacent. Then, since $k \geq 3$, there must be at least two non-adjacent vertices, x and y , neighboring v that are endpoints of edges in $\mathcal{M}$. But this is not possible, because $\{x, y, v, u, w\}$ induces a chair. ■

Theorem 2.2.2. *Let G be a chair-free graph and $\mathcal{M}$ be an induced matching of G . If there is some edge $uv = e \in E$ such that $|\mathcal{M} \cap C(e)| = k \geq 4$, then there is an induced G_k or an induced H_k in G .*

Proof. Clearly, $e \notin \mathcal{M}$ because $k \geq 4$. If u is an endpoint of some edge in $\mathcal{M}$, then v must be neighboring k edges of $\mathcal{M}$, and therefore by Lemma 2.2.2 there must be an induced G_k centered on v . Suppose now that e does not share any endpoint with edges from $\mathcal{M}$. Clearly, if u or v neighbors k edges from $\mathcal{M}$, then by Lemma 2.2.2 there is an induced G_k . Suppose that there is an edge $xy \in \mathcal{M}$ neighboring u but not v , and an edge $pq \in \mathcal{M}$ neighboring v but not u . Suppose w.l.o.g. that $xu \in E$ and $vq \in E$. It thus follows that $\{x, u, v, q\}$ induces a P_4 , which means that any endpoint of an edge in $\mathcal{M}$ (other than xy and pq) adjacent to u must be adjacent to v and vice versa (because G is chair-free). Therefore, there are $k - 2$ edges in $\mathcal{M}$ neighboring both, u and v . It follows that there are exactly $k - 1$ edges in $\mathcal{M}$ neighboring u and $k - 1$ edges in $\mathcal{M}$ neighboring v . Since $k - 1 \geq 3$, by Lemma 2.2.2 there is a G_{k-1} centered on u and a G_{k-1} centered on v . It follows that the vertex set of $\mathcal{M} \cap C(e)$ induces H_k . ■

Corollary 2.2.2. *If G is (G_k, H_k, chair) -free ($k \geq 4$), then the greedy algorithm has performance ratio $k - 1$.*

Corollary 2.2.3. *If G is (G_k, chair) -free ($k \geq 3$), then the greedy algorithm has performance ratio k .*

Corollary 2.2.4. *If G is $(K_{1,k}, \text{chair})$ -free ($k \geq 3$), then the greedy algorithm has performance ratio k .*

Table 2.1: Random graphs with $n = 100$ and $100 \leq m \leq 1000$.

Apx	OPT	OPT/Apx	LP/Apx	Δ
994.235	1313.783	1.321	1.981	16
586.682	1046.750	1.784	3.730	20
569.141	987.044	1.734	3.900	23
779.096	1071.847	1.376	2.735	20
531.683	979.944	1.843	4.280	29

2.3 Experimental results

Because of its simple structure and because we regard it as one of the main results of this thesis, it is of interest to analyze the practical behavior of Algorithm 1. The first thing to observe is that the linear program solved at the first iteration has a constraint matrix of size $|E| \times |E|$, which is the most expensive step in the computation. This imposes a practical limit on the size of the problem to be solved.

To our best knowledge there are no benchmark graphs in the literature for the induced matching problem. We thus explored the approximation behavior for randomly generated graphs, bipartite graphs and regular graphs. We acknowledge the experimental limitations of this approach. Unfortunately, we had no application at hand for IM to perform tests on some real-world instances.

The weights were picked randomly with a uniform distribution between 1 and 100. The results are shown in Tables 2.1, 2.2, 2.3, 2.4, and 2.5.

The gap between our solution values and the optimal ones is much smaller than the approximation factor. This suggests that the algorithm performs very good in practice, disregarding the gap with respect to the linear relaxation. It is worth noting that we computed the optimal values only for small instances, which means we cannot claim good behavior with respect to the optimal value for large instances. Furthermore, we observe that the denser the graph, the bigger is the integrality gap. This is expected to be so due to the local nature of the constraints in the linear program. Finally, we note that the gap with respect to the linear relaxation for random graphs is usually much smaller than Δ . However, as we have seen in Remark 2, there are instances where this gap is $\Delta - 1$.

Table 2.2: 20-regular graphs with $n = 40$.

Apx	OPT	OPT/Apx	LP/Apx	Δ
166.859	291.569	1.747	5.669	20
180.933	268.522	1.484	5.108	20
187.330	288.790	1.542	5.012	20
99.695	286.538	2.874	9.232	20
187.891	279.431	1.487	4.971	20

Table 2.3: 5-regular graphs with $n = 40$.

Apx	OPT	OPT/Apx	LP/Apx	Δ
490.066	625.157	1.276	1.548	5
418.108	628.044	1.502	1.905	5
436.627	568.391	1.302	1.724	5
459.120	654.537	1.426	1.717	5
336.008	587.630	1.749	2.294	5

Table 2.4: Bipartite graphs with $n_1 = 20$, $n_2 = 30$ and $Prob[uv \in E] = 0.2$.

Apx	OPT	OPT/Apx	LP/Apx	Δ
470.472	745.622	1.585	1.735	9
540.191	701.684	1.299	1.501	10
473.746	719.346	1.518	1.839	9
647.261	647.896	1.001	1.224	9
430.503	667.672	1.551	1.988	13

Table 2.5: Bipartite graphs with $n_1 = 20$, $n_2 = 30$ and $Prob[uv \in E] = 0.8$.

Apx	OPT	OPT/Apx	LP/Apx	Δ
175.597	287.360	1.636	6.260	29
196.746	287.731	1.462	5.588	27
184.185	268.497	1.458	5.802	27
185.272	352.590	1.903	5.929	28
198.288	266.653	1.345	5.671	29

Chapter 3

Clique Transversal Set

“The first principle is that you must not fool yourself and you are the easiest person to fool.”
—Richard Feynman

Let $G = (V, E)$ be a graph. A subset $S \subseteq V$ such that $C \cap S \neq \emptyset$ for every clique $C \in \mathcal{K}(G)$ is a *clique transversal set (CT-set)*. The *minimum CT-set (CTS)* problem is to find a CT-set of minimum cardinality. The *clique transversal number* $\tau_C(G)$ is the cardinality of such CT-set. The *weighted CT-set (WCTS)* problem is defined over vertex-weighted graphs, and the goal is to find a CT-set of G that minimizes its total weight.

In this chapter we present constant ratio approximation algorithms for the CTS problem for line, planar and bounded degree graphs. We adapt the ideas of Lee, Chang and Kloks [23, 64] to give a linear time algorithm for weighted graphs with bounded treewidth and a PTAS for WCTS on planar graphs.

3.1 Line graphs

In this section we give a 3-approximation algorithm for the CTS problem on unweighted line graphs and a 4-approximation algorithm for WCTS on line graphs.

In the following, we denote $L(G)$ as the input of the algorithms. Recall that due the algorithm of Roussopoulos we can find $G = (V, E)$ given $L(G)$ in linear time [79]. We assume that G has no isolated edges. Both algorithms described in this section consist in analyzing two types of cliques in $L(G)$: the *vertex-cliques* and *triangle-cliques*, where the former are given by $\delta(v) = \{e \in E : v \text{ is an endpoint of } e\}$ for some $v \in V$, and the latter are triangles in G that will correspond to cliques of size 3 in $L(G)$. See Figure 3.1.1 for an example. These are all the possible cliques in $L(G)$. Indeed, if K_k is clique of $L(G)$ with $k \geq 4$, then it must be associated to

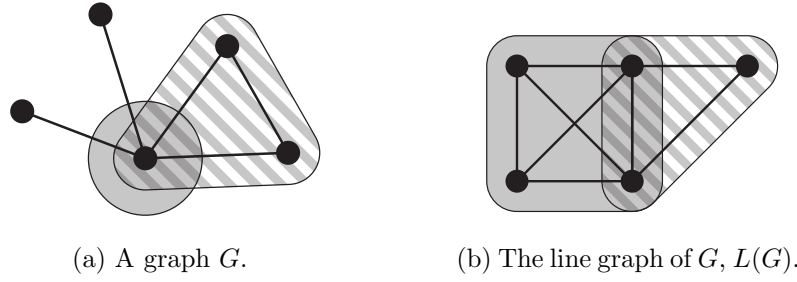


Figure 3.1.1: Clique types in line graphs: the vertex-clique has a solid shading and the triangle-clique a striped shading.

a vertex of degree k . If we have a triangle in $L(G)$ which is a clique, then it may be associated to a vertex of degree 3 in G or to a triangle in G . Finally, if we have a K_2 that it is a clique, it must correspond to a vertex of degree 2 in G whose neighbors are nonadjacent.

Let $S \subseteq V$ be the set of vertices that have degree at least 3 or have degree 2 but its neighbors are nonadjacent. For each $v \in S$ it is easy to see that $\delta(v)$ is a clique in $L(G)$. Furthermore, for any $z \in V \setminus S$ we have that $\delta(z)$ is not a clique. To see this, suppose $\deg(z) = 1$ and let u be the neighbor of z . Since we assumed G without isolated edges, u must have another neighbor; therefore $\delta(z) \subset \delta(u)$. If $\deg(z) = 2$, call x and y the neighbors of z (note that $xy \in E$ by assumption), and thus $\delta(z) \subset \{zx, zy, xy\}$, and the latter induces a complete subgraph of $L(G)$.

3.1.1 Unweighted case

Here we strongly exploit the fact that the graph is unweighted and the triangles have three edges. The algorithm consists of two phases: we first transverse all the triangle-cliques and afterwards extend this to transverse all the remaining vertex-cliques.

Start by computing a maximal edge-disjoint set of triangles in G , and take all the edges from these triangles as part of the solution, and call this set of edges F . Observe that we have $k = |F|/3$ edge-disjoint triangles. It is clear that we need at least one edge for each of the selected triangles, since they are edge-disjoint. Let $S' \subseteq S$ be the set of vertices associated to vertex-cliques that were yet not transversed by F . We now introduce a technical result before explaining the second phase of the algorithm.

Lemma 3.1.1. *Let $E^* \subseteq E$ be a minimum clique transversal of $L(G)$ and M a*

maximum matching in $G[S']$. Then,

$$|S'| - |M| \leq |E^* \setminus F|.$$

Proof. Let $E' \subseteq E$ of minimum cardinality that spans each vertex in S' (that is, every vertex of S' is an endpoint of some edge in E'). Consider the subgraph H with vertex set S' and the edges of E' that have both endpoints in S' . Let M' be a maximum matching in H . Since E' is minimal, it follows that $|S'| = 2|M'| + p$, where $p = |E' \setminus M'|$. On the other hand, we have $|M'| \leq |M|$ and $|E'| = |M'| + p$. Finally, $E^* \setminus F$ spans each vertex of S' , and then $|E'| \leq |E^* \setminus F|$. Therefore,

$$|S'| = |E'| + |M'| \leq |E^* \setminus F| + |M'| \leq |E^* \setminus F| + |M|.$$

■

In the second phase compute a maximum matching M in $G[S']$, and extend this edge subset in G in a greedy fashion until the subset spans all the vertices in S' . Name the resulting set E' . By definition we have

$$|E'| = |M| + |S'| - 2|M| = |S'| - |M|$$

and that $E' \cup F$ is a clique transversal for $L(G)$. Let E^* be an optimum clique transversal for $L(G)$. We have that

$$\begin{aligned} |E' \cup F| &= |S'| - |M| + |F| \\ &= |S'| - |M| + 3k \\ &\leq |E^* \setminus F| + 3|E^* \cap F| \\ &\leq 3(|E^* \setminus F| + |E^* \cap F|) \\ &= 3|E^*|. \end{aligned}$$

At this point, we have shown that this algorithm gives a clique transversal of size at most 3 times $\tau_C(L(G))$. Next, we show the algorithm runs in $O(n^{1.5} + m)$ time.

If n and m are the number of vertices and edges of $L(G)$ respectively, we can find G in $O(n + m)$ [79]. Furthermore, we can find all the triangles of G in $O(|E|^{1.5}) = O(n^{1.5})$ using [24]. Clearly, G has no isolated vertices. Therefore, G has n edges and $O(n)$ vertices; thus, we can find a maximum matching in G in

$O(\sqrt{|V|}|E|) = O(n^{1.5})$ using [73]. Hence, the resulting running time is $O(n^{1.5} + m)$.

In order to see the tightness of the analysis consider the following scenario. We build the graph G and make the analysis in G and not in $L(G)$. For any integer k take a path P_k , and for each vertex of the path add two neighbors so that a triangle is formed. The algorithm uses $3k$ edges, while an optimum solution uses only k edges, and this may be constructed by taking one edge of each triangle that it is incident to the vertex of P_k .

3.1.2 Weighted case

In this case we first transverse all the vertex-cliques of $L(G)$. To do so, we solve a minimum weighted edge cover in a graph resulting of adding a new vertex w to G and adding the edges uw for each $u \in V \setminus S$ with weight 0. It is easy to see that the optimum solution to this weighted edge cover has the same value as the best way of spanning the vertices in S . Furthermore, it is trivial to recover a solution to the original problem from the edge cover. Recall that weighted edge cover with nonnegative weights may be solved in polynomial time using the reduction to maximum matching given in [80, Theorem 27.2]. The weight of this edge cover is smaller than the weight of the optimum clique-transversal, since we covered only a subset of all the cliques.

The second part of the algorithm consists in spanning all the triangle-cliques of $L(G)$ that were not yet transversed. Denote this family $\mathcal{T}$. We are interested in selecting the smallest subset of edges of G that will span all the triangles from $\mathcal{T}$. This is a set cover problem, where the universe of objects is $\mathcal{T}$ and the subsets are $S_e = \{T \in \mathcal{T} : T \text{ contains } e\}$ for $e \in E$. Since each triangle appears exactly in three sets S_e , we can solve this set cover problem with an approximation factor of 3 using the well known frequency approximation algorithm given by Hochbaum in [52].

Taking the union of both sets of edges we get a feasible solution for clique transversal. The weight of edges in the resulting set is at most the sum of the weight of both parts of the algorithm, and thus the overall factor is 4.

3.2 A simple 3-approximation for planar graphs

In general, for a fixed integer t one can approximate the CTS problem with factor t on graphs where the maximum clique has size at most t . This may be achieved by taking a maximal set of vertex-disjoint cliques and outputting all their vertices.

This is a clique transversal because of the maximality of the set. On the other hand, each of these cliques requires at least one vertex in order to be transversal, and in our solution we use at most t vertices per clique. Therefore, we have a t -approximation algorithm. It is well known that any planar graph is K_5 -free (see, e.g., [63]); thus, the above approach leads to a 4-approximation algorithm. In the following, we improve this idea yielding a 3-approximation algorithm for planar graphs.

Begin by searching a maximal disjoint set of cliques of size 2 and 3, and use all their vertices as part of the solution, and call this vertex set S_1 . It is clear that all the cliques not yet transversal are of size 4. We look among these cliques a vertex v that belongs only to one non-transversal clique. To see that is always possible to find such vertex consider a planar representation of the graph, and assume for contradiction that there is no such vertex. Take any non-transversal clique of size 4. Clearly, one of its vertices is drawn inside the triangle induced by the rest of the vertices. Name this vertex u . Since there is no vertex that belongs to only one non-transversal clique, there must be another non-transversal clique of size 4 containing u . This clique has its own vertex in the middle which cannot be shared with the previous clique. We can apply this argument infinite times; thus, G has infinite vertices—a contradiction. Now, take any such vertex among the uncovered cliques and use the remaining 3 vertices of that clique as part of the solution. Iterate the same process with the remaining non-transversal cliques of size 4. Call the set of vertices added in the second phase S_2 . Clearly, $S_1 \cup S_2$ is a clique transversal. Furthermore, if in the first phase we considered k_1 disjoint cliques and in the second k_2 , then $k_1 + k_2 \leq \tau_C(G)$ and

$$|S_1 \cup S_2| = |S_1| + |S_2| \leq 3k_1 + 3k_2 \leq 3\tau_C(G).$$

Now we show this analysis to be tight. For any integer k consider the path P_{2k+1} , and for each edge of P_{2k+1} add a vertex adjacent to both of its endpoints. The algorithm may consider k disjoint triangles and use all its vertices as part of the solution, and thus the algorithm uses $3k$ vertices. On the other hand, we may construct a clique transversal of cardinality k by only taking half of the vertices from the initial path.

The presented algorithm is quite simple and fast in practice for unweighted planar graphs. However, approximation-wise we can do better by using Baker's technique. This approach was used for transversing all the *maximum* cliques in un-

weighted planar graphs [23]. In the following two sections we use a similar approach for transversing all *maximal* cliques of a weighted planar graph. We note that since we are dealing with weighted vertices we need to take a more careful approach than the presented in [23] when combining the solutions of slices. In the next section we give a linear time algorithm for bounded treewidth graphs for a slightly more general problem and in the following section we show how to use this algorithm for constructing a PTAS for weighted planar graphs.

3.3 Weighted bounded treewidth graphs

Consider the problem of transversing a given subset $\mathcal{L} \subseteq \mathcal{K}(G)$ of cliques in a vertex-weighted graph G with treewidth at most k . By the following lemma it follows that $|\mathcal{K}(G)|$ is $O(2^k n)$, and therefore one can encode the input $\mathcal{L}$ in linear space with respect to G .

Lemma 3.3.1 (Bodlaender and Möhring [14]). *For any tree decomposition of a graph G , each clique of G is contained in at least one node of the tree.*

Note that the WCTS problem is the instance where $\mathcal{L} = \mathcal{K}(G)$. In Lee's work $\mathcal{L}$ is defined implicitly as the set of cliques of *maximum* cardinality [64]. In the following paragraphs we replicate some of their ideas for completeness, adapting the proofs when necessary.

An $\mathcal{L}$ *simplicial augmentation* of a graph G is the graph resulting of adding to G one vertex per clique in $\mathcal{L}$, adding all the edges between that vertex and the vertices of the clique it represents, and removing all the vertices of G that do not appear in any clique of $\mathcal{L}$.

The following observation appears in [64], but here we give a constructive proof.

Lemma 3.3.2. *For a graph G of treewidth at most k and a set of cliques $\mathcal{L}$, the $\mathcal{L}$ simplicial augmentation of G has treewidth at most $k + 1$.*

Proof. Take any tree decomposition of G of width at most k . For each clique C in $\mathcal{L}$, take any node B in the decomposition that contains C (this exists because of Lemma 3.3.1). Add a new node neighboring with B with the vertices of C and the vertex associated to C in the $\mathcal{L}$ simplicial augmentation. Clearly, this new node has at most $\omega(G) + 1 \leq k + 2$ vertices. Now, from each node in this tree remove the vertices that do not appear in the augmentation. This new tree is a tree decomposition of width at most $k + 1$ for the $\mathcal{L}$ simplicial augmentation of G . ■

Observe, therefore, that once we build an $\mathcal{L}$ simplicial augmentation for G , we can obtain its tree decomposition in linear time. It is easy to prove the following lemma, similar statements were proved in [64] and [2].

Lemma 3.3.3. *A subset $S \subseteq V$ is a minimum weight transversal of the cliques in $\mathcal{L}$ if and only if S is a minimum weight dominating set of the $\mathcal{L}$ simplicial augmentation, where the weights of the vertices are the same as in G when possible and $+\infty$ for those introduced when building the augmentation.*

Theorem 3.3.1. *Given a vertex-weighted graph G with treewidth at most k and a set of cliques $\mathcal{L}$ of G , we can find a transversal of $\mathcal{L}$ of minimum weight in $O(4^k n)$ time.*

Proof. This follows from solving in $O(4^k n)$ time the weighted dominating set on the $\mathcal{L}$ simplicial augmentation of G using the algorithm given by Alber and Niedermeier in [1, Theorem 3.1]. ■

3.4 An approximation scheme for planar graphs

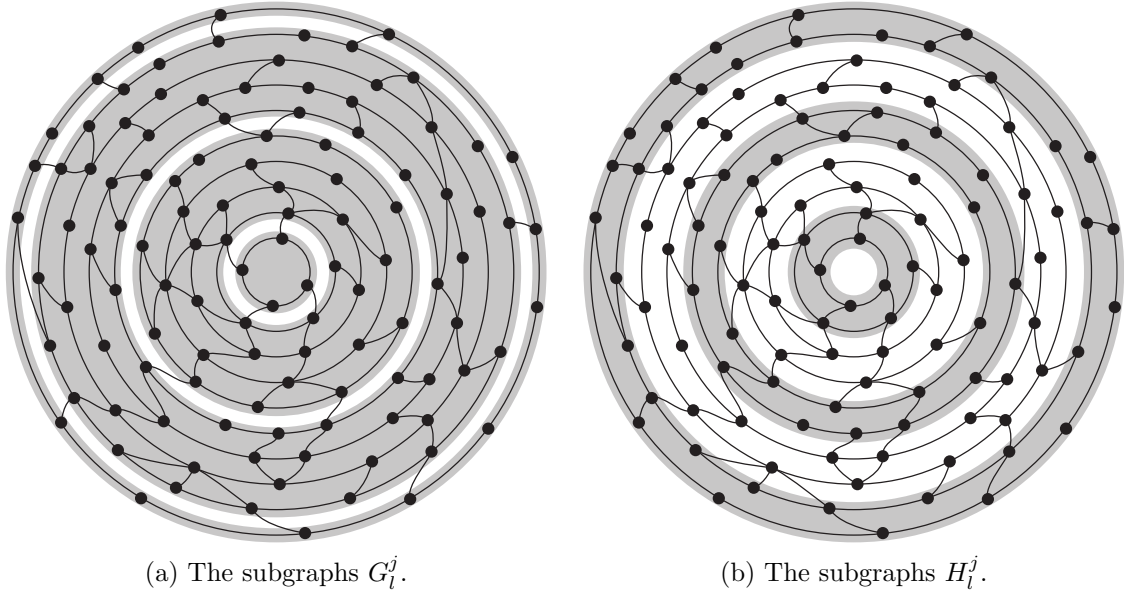
In the following we give a PTAS for WCTS on planar graphs using the algorithm explained in the previous section. Let G be a vertex-weighted plane graph.

Lemma 3.4.1. *Any clique C of G is either contained in one layer or in two consecutive layers.*

Proof. If C is contained in only one layer, then there is nothing to prove. Assume C has vertices in more than one layer. Let L_i be the layer with smallest index (i.e., the outermost) that intersects C . Since G is a plane graph, it follows that after removing $L_0 \cup \dots \cup L_i$ the remaining vertices of C must lie on the outer face of this graph since otherwise there must be two crossing edges each other, and thus L_{i+1} contains the rest of the vertices of C . ■

Assume a fixed $k \geq 2$. For each offset $l \in \{0, \dots, k-1\}$ and each $j \in \mathbb{Z}$, let $G_l^j = G[L_{l+jk} \cup \dots \cup L_{l+(j+1)k-1}]$ and $H_l^j = G[L_{l+jk-1} \cup L_{l+jk}]$. See Figure 3.4.1 for an example. Note that if r is the number of layers in G , then, for a fixed l , there are at most $O(r/k)$ non-empty graphs G_l^j and H_l^j , and therefore at most $O(r)$ graphs G_l^j and H_l^j are non-empty.

If we now were to compute a weighted clique transversal on each of these sub-graphs and output a union of these sets, we would not get any reasonable performance ratio. Consider for example the case when a maximal triangle has two

Figure 3.4.1: Decomposition for $l = 1$ and $k = 4$.

vertices u, v in G_l^j and one x in G_l^{j+1} , and there is no common neighbor to u and v in G_l^j . Suppose their weights are $w_u = w_v = 100$ and $w_x = 1$. Clearly, the above solution will pick u or v because uv is a clique of G_l^j , while it is clear that picking x for transversing this clique would be convenient.

In order to circumvent this problem we enforce transversing only the “original” cliques of G in each of these subgraphs. Formally, we are to transverse $\mathcal{K}(G) \cap \mathcal{K}(G_l^j)$ in G_l^j and $\mathcal{K}(G) \cap \mathcal{K}(H_l^j)$ in H_l^j . Let S_l^j be the computed solution for G_l^j ; and T_l^j the computed for H_l^j . Let $S_l = \bigcup_j S_l^j$ and let $T_l = \bigcup_j T_l^j$. The output of our algorithm is $S = S_l \cup T_l$ which minimizes its total weight over $l \in \{0, \dots, k-1\}$. Clearly, S is a feasible solution because by Lemma 3.4.1 every clique of G is contained in some of the subgraphs in the decomposition. We are now to prove the approximation ratio.

Let S^* be an optimal solution for G . For each $0 \leq l \leq k-1$, let $V_l = \bigcup_j V(H_l^j)$ be the vertices of the graphs with two layers in the decomposition for offset l . On the one hand, we have that

$$\sum_{l=0}^{k-1} w(V_l \cap S^*) = 2w(S^*),$$

and therefore there is some l such that

$$w(V_l \cap S^*) \leq \frac{2w(S^*)}{k}.$$

On the other hand, by definition each clique of $\mathcal{K}(G) \cap \mathcal{K}(G_l^j)$ has to be met by a vertex of G_l^j .

Therefore, we have that $w(S_l^j) \leq w(S^* \cap V(G_l^j))$, and thus by summing over j we get $w(S_l) \leq w(S^*)$. The same argument implies that $w(T_l) \leq w(V_l \cap S^*)$. It follows that there is some $0 \leq l \leq k-1$ such that

$$w(S) \leq w(S_l) + w(T_l) \leq w(S_l) + w(V_l \cap S^*) \leq w(S^*) + \frac{2w(S^*)}{k}.$$

Which implies the approximation ratio $1 + \frac{2}{k}$. We thus have the following.

Theorem 3.4.1. *There is a PTAS for the WCTS problem on planar graphs.*

Proof. For any $\epsilon > 0$ run the above algorithm with $k = \max \left\{ 2, \left\lceil \frac{2}{\epsilon} \right\rceil \right\}$. ■

3.5 Bounded degree graphs

In this section we introduce a $\left\lceil \frac{\Delta(G)+1}{2} \right\rceil$ -approximation algorithm for the CTS problem when restricted to bounded degree graphs. The algorithm consists in selecting a maximal vertex-disjoint set of *small* cliques, and uses all their vertices in the solution. If there are still non-transversed cliques, these must be *big*. In order to transverse these big cliques we take one of them and select a small subset of its vertices as part of the solution. As long as there are non-transversed cliques we apply the same procedure until every clique is transversed. The approximation ratio is yielded by the definition of *small* and a technical result.

Start by searching a maximal family $\mathcal{F}_1$ of disjoint cliques of size at most $\left\lceil \frac{\Delta(G)+1}{2} \right\rceil$. Denote V_1 to the vertices in the cliques of $\mathcal{F}_1$. Take V_1 as part of the solution. If all cliques were transversed, the algorithm ends here. Otherwise, let K be a non-transversed clique. By definition, $|K| > \left\lceil \frac{\Delta(G)+1}{2} \right\rceil$, and K is disjoint with all previously transversed cliques. Take any subset $S \subset K$ of size $\left\lceil \frac{\Delta(G)+1}{2} \right\rceil$ and add it to the solution.

Lemma 3.5.1. *Any non-transversed clique that intersects K must intersect S .*

Proof. For contradiction assume that there exists a non-transversed clique C that intersects K but does not intersect S . Since all the cliques of size at most $\left\lceil \frac{\Delta(G)+1}{2} \right\rceil$ were transversed, $|C| > \left\lceil \frac{\Delta(G)+1}{2} \right\rceil$. Take a vertex $v \in K \cap C$. The vertex v is adjacent to each vertex from S and C . Thus,

$$\deg(v) \geq |S| + |C| - 1 > 2 \left\lceil \frac{\Delta(G)+1}{2} \right\rceil - 1 \geq \Delta(G).$$

■

Once K is transversed apply the same procedure if there are still non-transversed cliques. Call the set of cliques used in the second phase of the algorithm $\mathcal{F}_2$ and the selected vertices of these cliques V_2 . Clearly, all these cliques are disjoint by the above lemma. Furthermore, $V_1 \cup V_2$ constitute a clique transversal set. The following holds:

$$|V_1 \cup V_2| = |V_1| + |V_2| \leq \left\lceil \frac{\Delta(G)+1}{2} \right\rceil (|\mathcal{F}_1| + |\mathcal{F}_2|) \leq \left\lceil \frac{\Delta(G)+1}{2} \right\rceil \tau_C(G).$$

The analysis is tight. To see this, it is enough to fix an integer k and consider a graph with two disjoint cliques, $G = K_{\lceil (k+1)/2 \rceil} \cup K_{k+1}$. Because of the larger clique we get $\Delta(G) = k$. On the other hand, the algorithm takes all the vertices from the smaller clique and $\left\lceil \frac{\Delta(G)+1}{2} \right\rceil$ from the bigger. Thus, the solution has $2 \left\lceil \frac{\Delta(G)+1}{2} \right\rceil$ vertices while $\tau_C(G) = 2$.

A possible linear-time implementation of this algorithm is the following. To build $\mathcal{F}_1$ initialize $\mathcal{F}_1 = \emptyset$, $V_1 = \emptyset$, and take any ordering $v_1, \dots, v_n$ of V . For $i = 1, \dots, n$ and $v_i \notin V_1$, consider all the subsets of $N(v_i) \cap \{v_{i+1}, \dots, v_n\} \cap \overline{V_1}$ of size at most $\left\lceil \frac{\Delta(G)+1}{2} \right\rceil - 1$. If T is one of these subsets and $T \cup \{v_i\}$ is a clique (clearly, v_i is the vertex of this clique with smallest index), then add this clique to $\mathcal{F}_1$ and its vertices to V_1 and continue with the next iteration. Observe that $|N(v_i) \cap \{v_{i+1}, \dots, v_n\} \cap \overline{V_1}| \leq \deg(v_i)$ and may be obtained in $O(\deg(v_i))$. Let $r_i = \min \left\{ \deg(v_i), \left\lceil \frac{\Delta(G)+1}{2} \right\rceil - 1 \right\}$. Thus, this procedure yields a running time of

$$\begin{aligned} O \left(\sum_{i=1}^n \left(\deg(v_i) + \sum_{j=1}^{r_i} \binom{\deg(v_i)}{j} (j^2 + j(\deg(v_i) - j)) \right) \right) &= O \left(\sum_{i=1}^n 2^{\deg(v_i)} \deg(v_i)^2 \right) \\ &= O(n). \end{aligned}$$

To build $\mathcal{F}_2$ the procedure is similar. We avoid considering the already trans-

versed cliques by searching big cliques in $N(v_i) \cap \{v_{i+1}, \dots, v_n\} \cap \overline{V_1} \cap \overline{V_2}$, for each $v_i \notin V_1 \cup V_2$. The selection of the subset S in each big clique is arbitrary. The running time of this phase is also $O(n)$.

Note that in [68], the authors gave a 1.96-approximation algorithm for cubic graphs. When relaxed to graphs with $\Delta(G) \leq 3$ our $\left\lceil \frac{\Delta(G)+1}{2} \right\rceil$ -approximation for bounded degree graphs yields a 2-approximation.

Chapter 4

Clique Independent Set

*“What seems to us as bitter
trials are often blessings in
disguise.”*

—Oscar Wilde

Let $G = (V, E)$ be a graph. A subset $\mathcal{S} \subseteq \mathcal{K}(G)$ such that $C \cap C' = \emptyset$ for any two different cliques C and C' in $\mathcal{S}$ is *clique independent (CI)*. The *maximum CI-set (CIS)* problem is to find a maximum cardinality CI-set. The *clique independence number* $\alpha_C(G)$ is the size of such CI-set. The *weighted CI-set (WCIS)* problem is defined over vertex-weighted graphs. Suppose the weights of the vertices are w_v for $v \in V$. For each clique $C \in \mathcal{K}(G)$ we let $w_C = \sum_{v \in C} w_v$ be the weight of C . The goal in the WCIS problem is to find a CI-set $\mathcal{S}$ that maximizes $\sum_{C \in \mathcal{S}} w_C$.

In this chapter we first give an approximation algorithm for CIS on line graphs in [Section 4.1](#). Next, we proceed to build a PTAS for the WCIS problem on planar graphs. For this purpose, in [Section 4.2](#) we give a linear time algorithm for a generalized version of WCIS problem on bounded treewidth graphs, and then in [Section 4.3](#) we use this result to build a PTAS for planar graphs.

4.1 Line graphs

As we already mentioned, we can obtain a graph G in linear time from a line graph L such that $L(G) = L$ using the algorithm of Roussopoulos [79]. Recall as well the discussion of [Section 3.1](#), where we defined the vertex-cliques and triangle-cliques of L . It is easy to see that a CI-set in L corresponds to a vertex-disjoint set of triangles in G and an independent set of vertices in $V(G)$ corresponding to vertex-cliques, such that no triangle shares a vertex with the independent set.

Suppose we are to compute an independent set for a triangle-free graph G . If we had a β -approximation for CI-set on line graphs, then we could use this algorithm to provide a β -approximation for $\alpha(G)$. To do so, add one neighbor to each vertex

of degree 1 in G , and let H be the resulting graph. It is easy to see that there is a natural 1-1 mapping from CI-sets in $L(H)$ and independent sets in G which preserves the sizes of the sets. Therefore, a β -approximate CI-set on $L(G)$ is a β -approximate independent set for G . This calls for the following question.

Question. Is it possible to approximate the independent set problem within a constant ratio on triangle-free graphs?

A negative answer would stand as well for CI-sets on line graphs by the above argument. To our best knowledge, no positive result nor an inapproximability result is known for this restriction of independent set. This motivates the search for algorithms with approximation ratios depending upon the maximum degree of the input graph.

Consider the approach of Algorithm 2 for computing a CI-set for a line graph L .

Algorithm 2 CIS(L) where L is a line graph

- 1: Compute G such that $L(G) = L$
 - 2: Compute a maximal set of vertex-disjoint triangles T of G
 - Let $W \subseteq V(G)$ be the vertices of G associated to the vertex-cliques of L
 - 3: Let $H = G[W \setminus V(T)]$, where $V(T)$ is the vertex set of the triangles in T
 - 4: Compute a β -approximate independent set I of H
 - 5: **return** $I \cup T$
-

Note that by definition $I \cup T$ corresponds to a CI-set in L . In what follows we analyze the performance ratio.

Let $S = S_V \cup S_T$ be an optimal CI-set for L , where S_V are the vertex-cliques and S_T the triangle-cliques in S . Let us abuse slightly the notation and refer with S_T to the corresponding triangles in G and with S_V to the corresponding vertices of G .

Lemma 4.1.1. $|S_T \cup (S_V \cap V(T))| \leq 3|T|$.

Proof. It follows from the maximality of T that any triangle in S_T has at least one vertex in $V(T)$. We use a “charging argument” for proving this result. Take any element $x \in S_T \cup (S_V \cap V(T))$ and “charge” it to some triangle in T with which x shares a vertex. Note that a triangle in T can be charged at most 3 times by this

procedure. Moreover, each element in $S_T \cup (S_V \cap V(T))$ is charged to some triangle in T . The result follows. $\blacksquare$

By definition of I we have that $\beta|I| \geq \alpha(H) \geq |S_V \setminus V(T)|$. Therefore,

$$\begin{aligned} |S| &= |S_T| + |S_V \cap V(T)| + |S_V \setminus V(T)| \\ &\leq 3|T| + \beta|I| \\ &\leq \max\{3, \beta\}(|T| + |I|). \end{aligned}$$

So far we have left out of the discussion the β -approximation of Line 4. In what follows we will relate $\Delta(H)$ with $\Delta(L)$ and suggest an approximation algorithm for independent set on bounded degree graphs.

Note that since L has degree at most $\Delta(L)$, then any edge $uv \in E(G)$ has at most $\Delta(L)$ neighboring edges. Moreover, if $uv \in E(H)$, then by definition of vertex-cliques it must be that $\deg_G(u) \geq 2$ and $\deg_G(v) \geq 2$. This means that if u is the vertex of maximum degree in H , then we have

$$\Delta(H) \leq \deg_G(u) + \deg_G(v) - 2 \leq \Delta(L).$$

Several approximation algorithms are known for the independent set problem with ratios depending on the maximum degree of the input graph. For an overview on this topic we suggest the survey by Halldorsson [49]. In particular, there is an approximation algorithm with ratio $\frac{\Delta(H)+3}{5} + \epsilon$ for a small $\epsilon > 0$ which runs in time $n^{O(1)}$ for a fixed $\epsilon > 0$ due to Berman et al. [6, 7]. Unfortunately, the running time polynomial appears to have prohibitive degree [49, 50]. Thus, in an implementation a larger β should be used. Nevertheless, the results of Berman et al. enable us to establish the following theoretical result.

Theorem 4.1.1. *For any $\epsilon > 0$ there is a $\max\{3, \frac{\Delta+3}{5} + \epsilon\}$ -approximation algorithm for CI-set on line graphs of degree at most Δ .*

Question. For any edge $uv \in E(H)$ we have that

$$\deg_H(u) + \deg_H(v) - 2 \leq \Delta(L).$$

We ask if some better approximation ratio may be achieved for such H . In a more generic way: is there a way improving the ratio $\frac{\Delta+3}{5} + \epsilon$ if any edge uv of

the input graph is such that $\deg(u) + \deg(v) \leq k$?

4.2 Weighted bounded treewidth graphs

In this section we present a linear time algorithm for a generalization of the WCIS problem on graphs with treewidth at most k , where k is a constant. In the definition of WCIS we gave before, we had a vertex-weighted graph as input; for convenience, in this section we use a more general formulation with a clique-weighted graph G as input, where each clique $C \in \mathcal{K}(G)$ has an associated weight $w_C \in \mathbb{Q}_{\geq 0}$. Note that since G has treewidth at most k , then G has at most $O(2^k n) = O(n)$ cliques.

The problem we solve in this section is the following: Given a subset $\mathcal{L} \subseteq \mathcal{K}(G)$ of cliques of G , we are to find a subset $\mathcal{S} \subseteq \mathcal{L}$ of maximum weight such that no two cliques of $\mathcal{S}$ share a vertex. Note that if we define $\mathcal{L} = \mathcal{K}(G)$ with $w_C = 1$ we get the WCIS problem. In order to make the dynamic programming algorithm we generalize further this problem.

For a node B and a subset of vertices $X \subseteq B$, we are to find a vertex-disjoint subset of cliques $\mathcal{S} \subseteq \mathcal{L} \cap \mathcal{K}(G_B)$ of maximum weight such that no clique in $\mathcal{S}$ has a vertex in $B \setminus X$. In other words, $V(\mathcal{S}) \cap (B \setminus X) = \emptyset$, where $V(\mathcal{S}) = \bigcup_{C \in \mathcal{S}} C$. Think of X as the available vertices from those in B . We denote this instance with $[B, X]$ (we omit writing $\mathcal{L}$ since it is defined only once).

Our algorithm will compute a table for each node B , which will have an entry $M[B, X]$ for each $X \subseteq B$ that will hold the optimum value for the problem $[B, X]$. Clearly, there will be at most 2^k entries for each node and the optimum value will be the one that is allowed to use any vertex of the root node of the decomposition. In the following we give the inductive rules for each node type of a nice tree decomposition.

- *Leaf.* B has exactly one vertex, and thus it is not a clique. We set

$$M[B, \emptyset] = M[B, B] = 0.$$

- *Join.* Assume B_1 and B_2 are the children of B . For each $X \subseteq B$ we set

$$M[B, X] = \max_{X = X_1 \dot{\cup} X_2} M[B_1, X_1] + M[B_2, X_2],$$

where $X_1 \dot{\cup} X_2$ is a partition of X .

- *Introduce.* Let B' be the child of B such that $B = B' \cup \{v\}$ and $v \notin B'$. For every $X \subseteq B'$ we set $M[B, X] = M[B', X]$, and for every $X \subseteq B$ with $v \in X$ we set

$$M[B, X] = \max \left\{ M[B', X \setminus \{v\}], \max_{\substack{C \in \mathcal{L}(X) \\ v \in C}} M[B', X \setminus C] + w_C \right\},$$

where $\mathcal{L}(X)$ is the subset of cliques of $\mathcal{L}$ contained in X .

- *Forget.* Let B' be the child of B such that $B' = B \cup \{v\}$ and $v \notin B$. We set $M[B, X] = M[B', X \cup \{v\}]$ for each $X \subseteq B$.

Theorem 4.2.1. *Let B be some node in the tree decomposition and $X \subseteq B$. Let $\mathcal{S}^* \subseteq \mathcal{L} \cap \mathcal{K}(G_B)$ be an optimal solution for $[B, X]$, then $M[B, X] = w(\mathcal{S}^*)$.*

Proof. The leaf case is trivial.

- *Forget.* Let $\mathcal{S}' \subseteq \mathcal{L}$ be the computed optimal solution to $[B', X \cup \{v\}]$. Note that by inductive hypothesis $w(\mathcal{S}') = M[B', X \cup \{v\}]$. Observe that $G_{B'} = G_B$ and $B \setminus X = B' \setminus (X \cup \{v\})$. Thus, on the one hand we have

$$V(\mathcal{S}^*) \cap (B' \setminus (X \cup \{v\})) = \emptyset,$$

which implies that $\mathcal{S}^*$ is feasible for $[B', X \cup \{v\}]$, and therefore $w(\mathcal{S}^*) \leq w(\mathcal{S}')$. On the other hand, $V(\mathcal{S}') \cap (B \setminus X) = \emptyset$, which means that $\mathcal{S}'$ is feasible for $[B, X]$, and thus $w(\mathcal{S}') \leq w(\mathcal{S}^*)$.

- *Introduce.* We first show that $w(\mathcal{S}^*) \leq M[B, X]$. We have the following two cases.

- $v \in V(\mathcal{S}^*)$. Then there is some clique $C \in \mathcal{S}^*$ such that $v \in C$. Since $v \notin B'$ and $C \in \mathcal{L}$, it follows that $C \subseteq B$. Moreover, $V(\mathcal{S}^*) \cap (B \setminus X) = \emptyset$ implies that $C \subseteq X$, and thus $C \in \mathcal{L}(X)$. Observe that $\mathcal{S}^* \setminus \{C\}$ is a feasible solution for $[B', X \setminus C]$ because

$$\begin{aligned} V(\mathcal{S}^* \setminus \{C\}) \cap (B' \setminus (X \setminus C)) &= (V(\mathcal{S}^*) \cap (B' \setminus (X \setminus C))) \setminus C \\ &= V(\mathcal{S}^*) \cap ((B' \setminus (X \setminus C)) \setminus C) \\ &= V(\mathcal{S}^*) \cap (B' \setminus X) \\ &= V(\mathcal{S}^*) \cap (B \setminus X) = \emptyset. \end{aligned}$$

Hence, $w(\mathcal{S}^* \setminus \{C\}) \leq M[B', X \setminus C]$, and thus

$$w(\mathcal{S}^*) \leq M[B', X \setminus C] + w_C \leq M[B, X].$$

- $v \notin V(\mathcal{S}^*)$. Clearly, $V(\mathcal{S}^*) \cap (B' \setminus (X \setminus \{v\})) = V(\mathcal{S}^*) \cap (B \setminus (X \setminus \{v\})) = \emptyset$. Furthermore, $V(\mathcal{S}^*) \subseteq V(G_{B'})$, which implies that $\mathcal{S}^*$ is feasible for $[B', X \setminus \{v\}]$. Hence, $w(\mathcal{S}^*) \leq M[B', X \setminus \{v\}] \leq M[B, X]$.

We now show that $w(\mathcal{S}^*) \geq M[B, X]$. Let $\mathcal{S}'$ be the solution returned by the algorithm, note that $\mathcal{S}' \subseteq \mathcal{K}(G_B) \cap \mathcal{L}$. There are two possible cases.

- $\mathcal{S}'$ is a solution to $[B', X \setminus \{v\}]$. Since $V(\mathcal{S}') \subseteq V(G_{B'})$, it follows that $v \notin V(\mathcal{S}')$. Then,

$$\begin{aligned} \emptyset &= V(\mathcal{S}') \cap (B' \setminus (X \setminus \{v\})) \\ &= V(\mathcal{S}') \cap ((B \setminus \{v\}) \setminus (X \setminus \{v\})) \\ &= V(\mathcal{S}') \cap (B \setminus X). \end{aligned}$$

Therefore, $\mathcal{S}'$ is feasible for $[B, X]$, and thus $w(\mathcal{S}^*) \geq w(\mathcal{S}') = M[B, X]$.

- $\mathcal{S}'$ is a solution to $[B', X \setminus C]$ for some $C \in \mathcal{L}(X)$ and $v \in C$. We will show that $\mathcal{S}' \cup \{C\}$ is feasible for $[B, X]$. The first thing to see is that this is a CI-set. Suppose that there is some $u \in V(\mathcal{S}') \cap C$. This means that there is some clique in $\mathcal{S}' \subseteq \mathcal{K}(G_{B'}) \cap \mathcal{L}$ that contains u , which implies that $u \neq v$. By definition, we have $V(\mathcal{S}') \cap (B' \setminus (X \setminus C)) = \emptyset$. But $u \in V(\mathcal{S}')$, $u \in B' = B \setminus \{v\}$, and $u \notin X \setminus C$ —a contradiction. Therefore, $\mathcal{S}' \cup \{C\}$ is a CI-set.

On the one hand, $C \cap (B \setminus X) = \emptyset$ because $C \subseteq \mathcal{L}(X)$. On the other hand, since $C \cap V(\mathcal{S}') = \emptyset$ and $v \notin V(\mathcal{S}')$, it follows that

$$\emptyset = V(\mathcal{S}') \cap (B' \setminus (X \setminus C)) = V(\mathcal{S}') \cap (B' \setminus X) = V(\mathcal{S}') \cap (B \setminus X).$$

Therefore, $V(\mathcal{S}' \cup \{C\}) \cap (B \setminus X) = \emptyset$. In other words, $\mathcal{S}' \cup \{C\}$ is feasible for $[B, X]$, and thus $w(\mathcal{S}^*) \geq w(\mathcal{S}') + w_C = M[B, X]$.

- *Join.* We first prove that $w(\mathcal{S}^*) \leq M[B, X]$. Let $\mathcal{S}_1 = \mathcal{S}^* \cap \mathcal{K}(G_{B_1})$ and $\mathcal{S}_2 = \mathcal{S}^* \setminus \mathcal{S}_1$. Let $X_1 = X \setminus V(\mathcal{S}_2)$ and $X_2 = X \setminus X_1$. In the following we show that $\mathcal{S}_1$ is feasible for $[B_1, X_1]$, and $\mathcal{S}_2$ for $[B_2, X_2]$.

We first prove the following technical claim:

$$\mathcal{K}(G_B) \cap \mathcal{L} = (\mathcal{K}(G_{B_1}) \cup \mathcal{K}(G_{B_2})) \cap \mathcal{L}.$$

On the one hand, if a clique C is in $\mathcal{K}(G_B)$, then there must be some node in the tree decomposition T^B containing the vertices of C . Since $B = B_1 = B_2$, one such node must be in T^{B_1} or in T^{B_2} . This implies that

$$\mathcal{K}(G_B) \subseteq \mathcal{K}(G_{B_1}) \cup \mathcal{K}(G_{B_2}),$$

and therefore $\mathcal{K}(G_B) \cap \mathcal{L} \subseteq (\mathcal{K}(G_{B_1}) \cup \mathcal{K}(G_{B_2})) \cap \mathcal{L}$. On the other hand, let C be a clique in $\mathcal{K}(G_{B_1}) \cap \mathcal{L}$ (or in $C \in \mathcal{K}(G_{B_2}) \cap \mathcal{L}$). Since $\mathcal{L} \subseteq \mathcal{K}(G)$ and $C \in \mathcal{L}$, it follows that C is a maximal clique of G . Since $C \in \mathcal{K}(G_{B_1})$, there is a complete subgraph in G_B that contains C , which cannot be extended by some vertex in V_{B_2} by the maximality of C in G . This implies the other contention.

Recall that we are to prove that $\mathcal{S}_1$ is feasible for $[B_1, X_1]$, and $\mathcal{S}_2$ for $[B_2, X_2]$. The first thing to note is that by definition $\mathcal{S}_1 \subseteq \mathcal{K}(G_{B_1})$. Since

$$\mathcal{K}(G_B) \cap \mathcal{L} = (\mathcal{K}(G_{B_1}) \cup \mathcal{K}(G_{B_2})) \cap \mathcal{L},$$

it follows that $\mathcal{S}_2 = \mathcal{S}^* \setminus \mathcal{S}_1 = \mathcal{S}^* \setminus \mathcal{K}(G_{B_1}) \subseteq \mathcal{K}(G_{B_2})$. It is trivial that $\mathcal{S}_1$ and $\mathcal{S}_2$ are CI-sets, both contained in $\mathcal{L}$. It remains to prove the relation of these proposed solutions with X_1 and X_2 . For $\mathcal{S}_1$ we have that

$$\begin{aligned} V(\mathcal{S}_1) \cap (B_1 \setminus X_1) &= V(\mathcal{S}_1) \cap (B \setminus (X \setminus V(\mathcal{S}_2))) \\ &= V(\mathcal{S}_1) \cap ((B \setminus X) \cup (B \cap V(\mathcal{S}_2))) \\ &= V(\mathcal{S}_1) \cap (B \setminus X) \\ &\subseteq V(\mathcal{S}^*) \cap (B \setminus X) \\ &= \emptyset. \end{aligned}$$

For $\mathcal{S}_2$ we have that

$$\begin{aligned} V(\mathcal{S}_2) \cap (B_2 \setminus X_2) &= V(\mathcal{S}_2) \cap (B \setminus (X \setminus X_1)) \\ &= V(\mathcal{S}_2) \cap (B \setminus (X \setminus (X \setminus V(\mathcal{S}_2)))) \\ &= V(\mathcal{S}_2) \cap (B \setminus (X \cap V(\mathcal{S}_2))) \end{aligned}$$

$$\begin{aligned}
&= V(\mathcal{S}_2) \cap (B \setminus X) \\
&\subseteq V(\mathcal{S}^*) \cap (B \setminus X) \\
&= \emptyset.
\end{aligned}$$

Since $X = X_1 \dot{\cup} X_2$ and $\mathcal{S}^* = \mathcal{S}_1 \dot{\cup} \mathcal{S}_2$, we conclude

$$w(\mathcal{S}^*) = w(\mathcal{S}_1) + w(\mathcal{S}_2) \leq M[B_1, X_1] + M[B_2, X_2] = M[B, X].$$

We now show that $w(\mathcal{S}^*) \geq M[B, X]$. For any partition $X_1 \dot{\cup} X_2 = X$, let $\mathcal{S}_1$ and $\mathcal{S}_2$ be some optimal solutions to $[B_1, X_1]$ and $[B_2, X_2]$, respectively. In the following we show that $\mathcal{S}_1 \cup \mathcal{S}_2$ is a feasible solution to $[B, X]$. Suppose it is not a CI-set, then there is some vertex $u \in C_1 \cap C_2$ for some $C_1 \in \mathcal{S}_1$ and $C_2 \in \mathcal{S}_2$. Since $C_1 \in \mathcal{K}(G_{B_1})$ and $C_2 \in \mathcal{K}(G_{B_2})$, it follows that $u \in B$. Since $V(\mathcal{S}_1) \cap (B_1 \setminus X_1) = \emptyset$, it follows that $u \in X_1$. Moreover, $u \in X_2$. This is a contradiction because $X_1 \cap X_2 = \emptyset$. Then $\mathcal{S}_1 \cup \mathcal{S}_2 \subseteq \mathcal{K}(G_B)$ is a CI-set, and both are in $\mathcal{L}$ by definition.

We have that $V(\mathcal{S}_1) \cap (B \setminus X) \subseteq V(\mathcal{S}_1) \cap (B_1 \setminus X_1) = \emptyset$, and in the same way $V(\mathcal{S}_2) \cap (B \setminus X) = \emptyset$. This implies that

$$V(\mathcal{S}_1 \cup \mathcal{S}_2) \cap (B \setminus X) = \emptyset.$$

Since this holds for any $X_1 \dot{\cup} X_2 = X$, it follows that $M[B, X] \leq w(\mathcal{S}^*)$. ■

Remark 3. Given some subset $\mathcal{L} \subseteq \mathcal{K}(G)$, we can compute the $M[\bullet, \bullet]$ table in linear time in a bottom-up fashion. The optimal solution to the generalized WCIS problem will be given by the entry $M[B_R, B_R]$ where B_R is the root node of the nice tree decomposition of G .

4.3 An approximation scheme for planar graphs

In the following we present a $(1 - \epsilon)$ -approximation algorithm for this problem using the algorithm given in the previous section.

For a fixed $k \geq 2$ and every offset $l \in \{0, \dots, k-1\}$, consider the graph G_l that results of removing the edges $E(L_i, L_{i+1})$ for each $i \equiv l \pmod k$, where L_j

($j \in \mathbb{Z}$) are the layers of G , and $E(L_i, L_{i+1})$ denotes the edges with one endpoint in L_i and the other in L_{i+1} . We can think of G_l as a union of slices G_l^j (using the notation of Section 3.4). Clearly, each G_l^j has at most k layers, and therefore it is a k -outerplanar graph, which in turn implies that its treewidth is bounded by a constant which depends on k . Let $\mathcal{C}(G_l^j) = \mathcal{K}(G) \cap \mathcal{K}(G_l^j)$ be the set of original cliques of G that are in G_l^j . For each l we run the algorithm described in the previous subsection on each G_l^j with $\mathcal{C}(G_l^j)$ as parameter, and we take the union of all these outputs. Name this solution $\mathcal{S}_l$. We output the set $\mathcal{S}_p$ that maximizes $\{w(\mathcal{S}_l) : l = 0, \dots, k-1\}$.

In what follows we prove that this is a $(1 - \frac{1}{k})$ -approximation. Clearly, each $\mathcal{S}_l$ is feasible because for each G_l^j we output a clique independent set, and since we removed $E(L_i, L_{i+1})$, no clique of $\mathcal{S}_l$ can span two different subgraphs G_l^j and $G_l^{j'}$. We are now to prove the factor.

Suppose $\mathcal{S}^*$ is an optimal solution. Let $\mathcal{C}_l \subseteq \mathcal{K}(G)$ be the set of cliques that intersect both, L_i and L_{i+1} , for $i \equiv l \pmod k$. We have that

$$\sum_{l=0}^{k-1} w(\mathcal{S}^* \cap \mathcal{C}_l) \leq w(\mathcal{S}^*),$$

and thus there must be some $l \in \{0, \dots, k-1\}$ such that

$$w(\mathcal{S}^* \cap \mathcal{C}_l) \leq \frac{w(\mathcal{S}^*)}{k}.$$

Therefore,

$$\begin{aligned} w(\mathcal{S}_p) &\geq w(\mathcal{S}_l) = \sum_j w(\mathcal{S}_l \cap \mathcal{C}(G_l^j)) \\ &\geq \sum_j w(\mathcal{S}^* \cap \mathcal{C}(G_l^j)) \\ &= w\left(\mathcal{S}^* \cap \bigcup_j \mathcal{C}(G_l^j)\right) \\ &= w(\mathcal{S}^* \setminus \mathcal{C}_l) && \text{Because } \mathcal{S}^* \setminus \mathcal{C}_l = \mathcal{S}^* \cap \bigcup_j \mathcal{C}(G_l^j) \\ &= w(\mathcal{S}^*) - w(\mathcal{S}^* \cap \mathcal{C}_l) \\ &\geq w(\mathcal{S}^*) \left(1 - \frac{1}{k}\right). \end{aligned}$$

Theorem 4.3.1. *There is a PTAS for the WCIS problem on planar graphs.*

Proof. For any $\epsilon > 0$ run the above algorithm with $k = \max \{2, \lceil \frac{1}{\epsilon} \rceil\}$. ■

Chapter 5

Neighborhood Covering Set

*“If you are going through hell,
keep going.”*

—Unknown¹

Let $G = (V, E)$ be a graph. A subset $S \subseteq V$ such that every edge $e \in E$ is in $E[v]$ for some $v \in S$ is *neighborhood covering (NC)*. The *minimum NC-set (NCS)* problem is to find a NC-set of minimum cardinality. The *weighted NC-set (WNCS)* problem is defined over vertex-weighted graphs and the goal is to find a NC-set that minimizes its total weight.

In [Section 5.1](#) we first give a linear time algorithm for a generalized version of WNCS problem on bounded treewidth graphs, and then in [Section 5.2](#) we use this result to build a PTAS for planar graphs. Finally, in [Section 5.3](#) we discuss the approximability of bounded degree graphs and line graphs.

5.1 Weighted bounded treewidth graphs

Given a graph $G = (V, E)$, we say that a vertex v *covers* an edge e if v is an endpoint of e or both endpoints of e are neighbors of v . Given a subset $X \subseteq V$, we define

$$\text{cov}(X) = \{uv \in E : \exists x \in X \text{ where } u, v \in N[x]\} = \bigcup_{x \in X} E[x]$$

to be the edges of G covered by vertices in X . If U is a vertex set we let $E(U)$ denote the set of edges with both endpoints in U .

Let T be a nice tree decomposition of G . Given a node B , a set of vertices $X \subseteq B$ and a set of edges F we define the problem $[B, X, F]$, a generalization of WNCS, as follows. We must pick exactly X from B into the solution, and we are not required to cover the edges in F —think of these as those already covered by

¹This quote seems to be wrongly attributed to Sir Winston Churchill.

vertices chosen higher up in the tree decomposition. We must cover the remaining edges of G_B . The objective is to minimize the total weight of the chosen set of vertices. More formally, $[B, X, F]$ asks for a minimum weight set $O \subseteq V_B$ such that $E_B \setminus F \subseteq \text{cov}(O)$ and $O \cap B = X$. For efficiency reasons we assume that $F \subseteq E(B)$. Note that unlike the original WNCS problem, this modified version may have some unfeasible instances, for which we define the optimum value to be $+\infty$.

The algorithm computes a solution using memoization in a top-down approach. We use $[B, X, F]$ to denote the problem and the states of the dynamic program. We let $M[B, X, F]$ denote the computed value for state $[B, X, F]$. Assume the recursion is at state $[B, X, F]$, we proceed as follows depending upon the type of the node B .

- *Leaf.* Set $M[\{v\}, X, F] = \begin{cases} w_v & \text{if } v \in X, \\ 0 & \text{if } v \notin X. \end{cases}$
- *Forget.* Suppose $v \notin B$ and $B' = B \cup \{v\}$ is the child of B . We set

$$M[B, X, F] = \min\{M[B', X, F_1], M[B', X \cup \{v\}, F_2]\},$$

where $F_1 = (F \setminus \text{cov}(X)) \cap E_{B'}$ and $F_2 = (F \setminus \text{cov}(X \cup \{v\})) \cap E_{B'}$.

- *Introduce.* Suppose B' is the child of B , $v \notin B'$ and $B = B' \cup \{v\}$. We set

$$M[B, X, F] = \begin{cases} M[B', X \setminus \{v\}, F'] + w_v & \text{if } v \in X \text{ and} \\ & \forall uv \in E(B) : uv \in \text{cov}(X) \cup F, \\ M[B', X, F'] & \text{if } v \notin X \text{ and} \\ & \forall uv \in E(B) : uv \in \text{cov}(X) \cup F, \\ +\infty & \text{otherwise,} \end{cases}$$

where $F' = ((F \cup \text{cov}(X)) \setminus \text{cov}(X \setminus \{v\})) \cap E_{B'}$.

- *Join.* Suppose B_1 and B_2 are the children of B . We set

$$M[B, X, F] = \min_{E' \subseteq E(B) \setminus F} M[B_1, X, F_1] + M[B_2, X, F_2] - w(X),$$

where

$$\begin{aligned} F_1 &= ((F \cup E') \setminus \text{cov}(X)) \cap E_{B_1}, \\ F_2 &= ((F \cup \overline{E'}) \setminus \text{cov}(X)) \cap E_{B_2}, \end{aligned}$$

$$\overline{E'} = E(B) \setminus (F \cup E').$$

The intuition behind this rule is that we split the responsibility between B_1 and B_2 for covering the edges not yet covered. We need to guess which splitting of responsibilities is the best one, and thus we need to take any possible E' as defined above.

We first prove that the recursive rules enforce the invariant $F \subseteq E(B)$ throughout all reached states. Then, in Theorem 5.1.1 we prove that these rules yield the optimal solution.

Lemma 5.1.1. *If the computation begins at $[B, X, F]$ such that $F \subseteq E(B)$, then for every reached state $[B', X', F']$ it holds that $F' \subseteq E(B')$.*

Proof. We prove by induction in the tree decomposition. The statement clearly holds if the state $[B', X', F']$ is the state on which the computation begins. Suppose $[B', X', F']$ is not the root state, then it must have some parent or invoking state $[\tilde{B}, \tilde{X}, \tilde{F}]$ such that $\tilde{F} \subseteq E(\tilde{B})$ by inductive hypothesis. The following are the possible cases for $\tilde{B}$.

- *Forget.* We have that $\tilde{B} \cup \{v\} = B'$, $v \notin \tilde{B}$ and

$$\begin{aligned} F_1 &= (\tilde{F} \setminus \text{cov}(\tilde{X})) \cap E_{B'} \subseteq \tilde{F} \subseteq E(\tilde{B}) \subseteq E(B'), \\ F_2 &= (\tilde{F} \setminus \text{cov}(\tilde{X} \cup \{v\})) \cap E_{B'} \subseteq \tilde{F} \subseteq E(\tilde{B}) \subseteq E(B'). \end{aligned}$$

Since $[B', X', F']$ is one of the two children of the state $[\tilde{B}, \tilde{X}, \tilde{F}]$ (i.e., either $[B', \tilde{X}, F_1]$ or $[B', \tilde{X} \cup \{v\}, F_2]$), the result follows.

- *Introduce.* We have that $\tilde{B} = B' \cup \{v\}$ and $v \notin B'$. By definition,

$$\begin{aligned} F' &= ((\tilde{F} \cup \text{cov}(\tilde{X})) \setminus \text{cov}(\tilde{X} \setminus \{v\})) \cap E_{B'} \\ &= ((\tilde{F} \setminus \text{cov}(\tilde{X} \setminus \{v\})) \cup (\text{cov}(\tilde{X}) \setminus \text{cov}(\tilde{X} \setminus \{v\}))) \cap E_{B'}. \end{aligned}$$

Observe that $\tilde{F} \setminus \text{cov}(\tilde{X} \setminus \{v\}) \subseteq E(\tilde{B})$ and $E(\tilde{B}) \cap E_{B'} = E(\tilde{B} \setminus \delta(v)) = E(B')$. It remains to show that $(\text{cov}(\tilde{X}) \setminus \text{cov}(\tilde{X} \setminus \{v\})) \cap E_{B'} \subseteq E(B')$. Take an edge $xy \in (\text{cov}(\tilde{X}) \setminus \text{cov}(\tilde{X} \setminus \{v\})) \cap E_{B'}$ and suppose for contradiction that $xy \notin E(B')$. Since $xy \in \text{cov}(\tilde{X}) \setminus \text{cov}(\tilde{X} \setminus \{v\})$, xy must be covered by v . Since $xy \in E_{B'}$ and $v \notin V_{B'}$, it follows that $x \neq v$, $y \neq v$, and $\{vy, vx\} \subseteq E_{\tilde{B}}$. Since vxy is a triangle and $T^{\tilde{B}}$ a tree decomposition, there must be a node

B'' in $T^{\tilde{B}}$ such that $\{v, x, y\} \subseteq B''$. Since $xy \in E_{B'}$, there must be a node B''' in $T^{B'}$ that contains x and y , and since $B'' \notin T^{B'}$ and every node lying in the path from B''' to B'' must contain x and y , it must be the case that $E(B')$ contains xy because B' lies in this path.

- *Join.* The result follows directly from the fact that $E' \subseteq E(\tilde{B}) = E(B_1)$ and $\overline{E'} \subseteq E(\tilde{B}) = E(B_2)$.

■

Theorem 5.1.1. *The value $M[B, X, F]$ computed using the above rules is the value of an optimum solution to $[B, X, F]$ if it is feasible, and $+\infty$ otherwise.*

Proof. By induction in the tree decomposition.

- *Leaf.* Trivial.
- *Forget.* Throughout this part of the proof we use several times the fact that $E_B = E_{B'}$ if B is a Forget node and B' its child.

It is easy to prove that $[B, X, F]$ is feasible if and only if $[B', X, F_1]$ or $[B', X \cup \{v\}, F_2]$ is feasible. We now assume that $[B, X, F]$ is feasible.

Let S^* be an optimal solution to the instance $[B, X, F]$. We first prove that $w(S^*) \leq M[B, X, F]$.

Suppose $[B', X, F_1]$ is feasible and let S' be an optimal solution to $[B', X, F_1]$, we prove that S' is feasible for $[B, X, F]$. First note that $S' \cap B = S' \cap B' = X$ because $v \notin S'$. Furthermore, S' covers the edges of

$$\begin{aligned} E_{B'} \setminus F_1 &= E_{B'} \setminus ((F \setminus \text{cov}(X)) \cap E_{B'}) \\ &= E_B \setminus (F \setminus \text{cov}(X)). \end{aligned}$$

Since $F \setminus \text{cov}(X) \subseteq F$, it follows that $E_B \setminus F \subseteq E_B \setminus (F \setminus \text{cov}(X))$, which implies that S' covers $E_B \setminus F$, and thus S' is feasible for $[B, X, F]$.

Suppose now $[B', X \cup \{v\}, F_2]$ is feasible and let S' be an optimal solution to $[B', X \cup \{v\}, F_2]$, we prove that S' is feasible for $[B, X, F]$. Observe that $S' \cap B = S' \cap (B' \setminus \{v\}) = X$. Let us show that every edge in $E_B \setminus F$ is covered by S' . By definition, S' covers

$$E_{B'} \setminus F_2 = E_{B'} \setminus ((F \setminus \text{cov}(X \cup \{v\})) \cap E_{B'})$$

$$= E_B \setminus (F \setminus \text{cov}(X \cup \{v\})).$$

Since $F \setminus \text{cov}(X \cup \{v\}) \subseteq F$, it follows that $E_B \setminus F \subseteq E_B \setminus (F \setminus \text{cov}(X \cup \{v\}))$, which implies that S' covers $E_B \setminus F$, and thus it is feasible for $[B, X, F]$. We have thus proven $w(S^*) \leq M[B, X, F]$.

We now prove that $w(S^*) \geq M[B, X, F]$. For this purpose, we first show that if $v \notin S^*$, then S^* is feasible for $[B', X, F_1]$, and if $v \in S^*$, then S^* is feasible for $[B', X \cup \{v\}, F_2]$.

If $v \notin S^*$, then $B' \cap S^* = (B \cup \{v\}) \cap S^* = X$. Furthermore,

$$\begin{aligned} E_{B'} \setminus F_1 &= E_{B'} \setminus ((F \setminus \text{cov}(X)) \cap E_{B'}) \\ &= E_B \setminus (F \setminus \text{cov}(X)) \\ &= (E_B \setminus F) \cup (E_B \cap \text{cov}(X)), \end{aligned}$$

where the first term of the union is covered by S^* by definition; and the second one because $X \subseteq S^*$. This implies that if $v \notin S^*$, then S^* is feasible for $[B', X, F_1]$.

If $v \in S^*$, then by definition we have $S^* \cap B' = X \cup \{v\}$. Furthermore,

$$\begin{aligned} E_{B'} \setminus F_2 &= E_{B'} \setminus ((F \setminus \text{cov}(X \cup \{v\})) \cap E_{B'}) \\ &= E_B \setminus (F \setminus \text{cov}(X \cup \{v\})) \\ &= (E_B \setminus F) \cup (E_B \cap \text{cov}(X \cup \{v\})), \end{aligned}$$

where the first term is covered by definition; and the second one because $X \cup \{v\} \subseteq S^*$. Thus, $w(S^*) \geq M[B, X, F]$.

- *Introduce.* In what follows we will prove that $[B, X, F]$ admits a feasible solution if and only if $[B', X \setminus \{v\}, F']$ is feasible and $uv \in \text{cov}(X) \cup F$ for each $uv \in E(B)$. It will follow from the proof that the optimum value for $[B, X, F]$ is $M[B, X, F]$.

We begin showing the *only if* part. Let S^* be an optimal feasible solution to $[B, X, F]$. Suppose for contradiction that there is an edge $uv \in E(B)$ such that $uv \notin \text{cov}(X) \cup F$, that is, not covered by X but which is mandatory to cover since $uv \notin F$. Then, there must be some vertex $w \in S^* \setminus X$ such that $uw, vw \in E_B$. Since T^B is a tree decomposition, there must be some node B''

in T^B that contains the triangle uvw , which is not possible because $w \notin B$ and $v \notin B'$.

It remains to see that $[B', X \setminus \{v\}, F']$ is feasible. We claim that $S^* \setminus \{v\}$ is a feasible solution to this problem.

First note that $(S^* \setminus \{v\}) \cap B' = X \setminus \{v\}$. To see that $S^* \setminus \{v\}$ covers $E_{B'} \setminus F'$, take any edge $xy \in E_{B'}$ and consider the following three cases.

- $xy \in E[u]$ for some $u \in S^* \setminus \{v\}$. Then clearly $xy \in \text{cov}(S^* \setminus \{v\})$.
- $xy \in E[v]$ and $xy \notin E[u]$ for any $u \in S^* \setminus \{v\}$. Then, $xy \notin \text{cov}(X \setminus \{v\})$, and therefore $xy \in F'$.
- $xy \notin \text{cov}(S^*)$. Since S^* is feasible for $[B, X, F]$, it must be the case that $xy \in F$. Moreover, $xy \notin \text{cov}(X \setminus \{v\})$, and therefore $xy \in F'$.

Which proves that $xy \in \text{cov}(S^* \setminus \{v\})$ for any $xy \in E_{B'} \setminus F'$.

Observe that this feasibility relation implies that $w(S^*) \geq M[B', X, F']$ if $v \notin X$ and $w(S^*) \geq M[B', X \setminus \{v\}, F'] + w_v$ if $v \in X$. In other words $w(S^*) \geq M[B, X, F]$.

We now prove the *if* part. Let S' be an optimal solution to $[B', X \setminus \{v\}, F']$. Consider the following two cases.

- $v \notin X$. In the following we show that S' is feasible for $[B, X, F]$. First observe that $S' \cap B = S' \cap (B' \cup \{v\}) = S' \cap B' = X \setminus \{v\} = X$. Suppose that there is an edge $xy \in E_B \setminus F$ not covered by S' . Since $v \notin X$, we have $F' = (F \setminus \text{cov}(X)) \cap E_{B'} \subseteq F$, and therefore $E_B \setminus F \subseteq E_B \setminus F'$. It follows that $xy \in E_B \setminus F'$, and therefore $xy \in E_B \setminus E_{B'}$. Hence, it must be the case that either $x = v$ or $y = v$. Since $uv \in \text{cov}(X) \cup F$ for each $uv \in E(B)$, it follows that $xy \in \text{cov}(X) \cup F \subseteq \text{cov}(S') \cup F$, which is a contradiction. Therefore, S' is feasible for $[B, X, F]$, which implies that $w(S^*) \leq w(S') = M[B, X, F]$ for some optimal solution S^* of $[B, X, F]$.
- $v \in X$. In the following we prove that $\tilde{S} = S' \cup \{v\}$ is a feasible solution to $[B, X, F]$. First note that

$$\tilde{S} \cap B = (S' \cup \{v\}) \cap (B' \cup \{v\}) = (S' \cap B') \cup \{v\} = X.$$

Now we show that each $xy \in E_B \setminus F$ is covered by $\tilde{S}$. If $x = v$ or $y = v$, then xy is clearly covered by $\tilde{S}$. Assume $x \neq v$, $y \neq v$, and

for contradiction suppose xy is not covered by $\tilde{S}$. Since $xy \in E_B$ and $xy \notin \delta(v)$, it follows that $xy \in E_{B'}$. Recall that $S' \subseteq \tilde{S}$ and S' covers $E_{B'} \setminus F'$. Therefore, $xy \notin E_{B'} \setminus F'$, and thus $xy \in F'$, which implies that $xy \in F \cup \text{cov}(X)$. If it is the case that $xy \in F$, then $xy \notin E_B \setminus F$, which is a contradiction. If it is the case that $xy \in \text{cov}(X)$, then xy is covered by $X \subseteq \tilde{S}$ —another contradiction. We conclude that $\tilde{S}$ covers $E_B \setminus F$, and therefore $\tilde{S}$ is feasible for $[B, X, F]$. It follows that $w(\tilde{S}) = M[B', X \setminus \{v\}, F'] + w_v \geq w(S^*)$ for some optimal solution of $[B, X, F]$.

Combining both cases we obtain $w(S^*) \geq M[B, X, F]$ if $[B, X, F]$ admits a feasible solution.

We thus conclude that $M[B, X, F]$ is the optimal value for the problem $[B, X, F]$ if such exists, or $+\infty$ otherwise.

- *Join.* It is easy to see that $[B, X, F]$ admits a feasible solution if and only if there is some $E' \subseteq E(B) \setminus F$ such that $[B_1, X, E_1]$ and $[B_2, X, E_2]$ admit feasible solutions. We assume this in what follows.

Let S^* be an optimal solution to the instance $[B, X, F]$. We first show that $w(S^*) \geq M[B, X, F]$. Let $S_1 = S^* \cap V_{B_1}$, $S_2 = S^* \cap V_{B_2}$ and

$$E' = (E(B) \cap \text{cov}(S_2)) \setminus F. \quad (5.1.1)$$

We claim that S_1 is feasible for $[B_1, X, F_1]$ and S_2 for $[B_2, X, F_2]$ (note that F_1 and F_2 depend of E').

- S_1 is feasible for $[B_1, X, F_1]$. The first thing to note is that $S_1 \cap B_1 = X$ and $S_1 \subseteq V_{B_1}$ by definition of S_1 . Let us prove that S_1 covers $uv \in E_{B_1} \setminus F_1$. Since $uv \notin F_1$, it must be because $uv \in \text{cov}(X)$ or $uv \notin F \cup E'$. In the former case there is nothing to prove. For the latter, we can furthermore assume $uv \notin \text{cov}(X)$. Since $uv \notin F$ and S^* is a feasible solution for $[B, X, F]$, then S^* covers uv , which means that there is some $w \in S^*$ such that $uv \in E[w]$. If $w \in S_1$, then there is nothing to prove. Suppose $w \in S^* \setminus S_1$. Therefore, $w \in S_2$, which means that $uv \in \text{cov}(S_2)$. Observe that if $uv \notin E(B)$, then without losing overall generality (w.l.o.g.) $v \notin B$, and therefore $w \in B$, which implies $w \in X$, and thus $w \in S_1$ —which is a contradiction. Therefore, $uv \in E(B)$, which implies that $uv \in E'$ by

definition of E' , which is a contradiction because $uv \notin F \cup E'$. We have thus seen that $uv \in \text{cov}(S_1)$, and thus S_1 covers $E_{B_1} \setminus F_1$.

- S_2 is feasible for $[B_2, X, F_2]$. Note that $S_2 \cap B_2 = X$ and $S_2 \subseteq V_{B_2}$. Let us prove that S_2 covers $uv \in E_{B_2} \setminus F_2$. Since $uv \notin F_2$, it must be the case that $uv \in \text{cov}(X)$ or $uv \notin F \cup \overline{E'}$. In the former case there is nothing to prove. For the latter, we can assume that $uv \notin \text{cov}(X)$. Since $uv \notin F$ and S^* covers $E_B \setminus F$, it follows that there is some $w \in S^*$ that covers uv . If $w \in S_2$, then there is nothing to prove. Suppose no vertex of S_2 covers uv , which means that $w \in V_{B_1} \setminus B$. Since T^B is a tree decomposition and w covers uv , it must be the case that $uv \in E(B)$. Since S_2 does not cover uv and $uv \in E(B) \setminus F$, it follows that $uv \in \overline{E'}$ (by the definition of E' in 5.1.1), and this in turn implies that $uv \in F_2$ —a contradiction. Hence S_2 covers $E_{B_2} \setminus F_2$.

It follows from these two cases that

$$\begin{aligned} w(S^*) &= w(S_1) + w(S_2) - w(X) \\ &\geq \min_{E' \subseteq E(B) \setminus F} M[B_1, X, F_1] + M[B_2, X, F_2] - w(X) \\ &= M[B, X, F]. \end{aligned}$$

We now prove $w(S^*) \leq M[B, X, F]$. Let $E' \subseteq E(B) \setminus F$ that minimizes $M[B_1, X, F_1] + M[B_2, X, F_1]$. Let S_1 and S_2 be the optimal solutions for $[B_1, X, F_1]$ and $[B_2, X, F_2]$, respectively. In what follows we prove that the set $S' = S_1 \cup S_2$ is feasible for $[B, X, F]$. First note that $S' \cap B = S' \cap B_1 = X$ and $S' \subseteq V_B$. Let us show that S' covers every $xy \in E_B \setminus F$. We have the following two cases.

- $xy \in E(B)$. Suppose $xy \in E'$. Since S_2 covers $E_{B_2} \setminus F_2$ and $E(B) \subseteq E_{B_2}$, it follows that S_2 covers

$$\begin{aligned} E(B) \setminus ((F \cup \overline{E'}) \setminus \text{cov}(X)) &= (E(B) \cap \text{cov}(X)) \cup (E(B) \setminus (F \cup \overline{E'})) \\ &\supseteq E(B) \setminus (F \cup \overline{E'}). \end{aligned}$$

Since $xy \notin F$ and $xy \notin E(B) \setminus (F \cup \overline{E'}) = \overline{E'}$, it follows that S_2 covers xy .

Suppose now that $xy \notin E'$, that is $xy \in \overline{E'}$. Since S_1 covers $E_{B_1} \setminus F_1$ and $E(B) \subseteq E_{B_1}$, we have that S_1 covers

$$E(B) \setminus ((F \cup E') \setminus \text{cov}(X)) = (E(B) \cap \text{cov}(X)) \cup (E(B) \setminus (F \cup E')).$$

Since $xy \notin F \cup E'$, it follows that S_1 covers xy .

- $xy \notin E(B)$. Since $xy \in E_B$ and $xy \notin E(B)$, we can assume w.l.o.g. that $x \notin B$. It cannot be the case that $x \in V_{B_1}$ and $x \in V_{B_2}$ at the same time. If $x \in V_{B_1}$, then there must be a node B' of T^{B_1} such that $xy \in E(B')$. Since S_1 covers $E_{B_1} \setminus F_1$, and $xy \notin F_1 = ((F \cup E') \setminus \text{cov}(X)) \cap E_{B_1}$ because $E' \subseteq E(B)$ and $xy \notin F$, it follows that xy is covered by S_1 . If $x \in V_{B_2}$, then $xy \in B'$ for some node B' of T^{B_2} . As before, S_2 covers $E_{B_2} \setminus F_2$ and $xy \notin F_2 = ((F \cup \overline{E'}) \setminus \text{cov}(X)) \cap E_{B_2}$, and therefore xy is covered by S_2 .

We have seen that $w(S^*) \leq w(S') = w(S_1) + w(S_2) - w(X) = M[B, X, F]$. Hence $w(S^*) = M[B, X, F]$.

■

In the following we show that the above algorithm admits a linear time implementation. The first observation is that we cannot have an explicit adjacency matrix for G . In other words, it is not obvious how to check adjacencies within a node B in constant time. As to solve this issue we propose having an adjacency matrix of size $|B| \times |B|$ for each node B of the tree decomposition. Clearly, computing such matrix for a fixed node is $O(n + m)$, which means that this construction has to be done in a more clever way than paying $O(n + m)$ for each node of the decomposition. One way of doing this is by using the computed matrix of the parent node, and paying $O(n + m)$ only at the root node. Clearly, for introduce, leaf and join nodes this is $O(1)$. If the node is a forget node, then we need to add a new entry to the matrix we already have and check adjacencies for each vertex in the parent node. This can be done in $O(\deg(v))$, where v is the vertex added in the forget node. The key observation is that each vertex v can be added into the tree decomposition in a forget node at most once, and therefore the time for computing all the forget cases is $O(m)$. Hence, this construction requires $O(n + m)$ at the root node, and $O(n + m)$ for the rest of the nodes since the number of nodes is $O(n)$. This yields the following.

Lemma 5.1.2. *Let G be a graph of bounded treewidth. Given a nice tree decomposition for G of size $O(n)$ with nodes $B_1, \dots, B_s$, one can compute the set $\{A_1, \dots, A_s\}$ in $O(\text{poly}(\text{tw}(G)) \cdot (n + m))$ time, where A_i is the adjacency matrix for $G[B_i]$ ($1 \leq i \leq s$) and $\text{poly}(\text{tw}(G))$ is some polynomial on $\text{tw}(G)$.*

Theorem 5.1.2. *Assuming the representation of the above lemma, the running time for computing $M[B, X, F]$ is $O(\text{poly}(\text{tw}(G)) \cdot n2^{5\text{tw}(G)})$.*

Proof. On the one hand, for every node B' reached in the computation there are at most $2^{\text{tw}(G)+1}$ subsets $X' \subseteq B'$, and for every $X' \subseteq B'$ there are at most $O((2^{\text{tw}(G)+1})^2) = O(2^{2\text{tw}(G)})$ subsets $F' \subseteq E(B')$. Since the tree decomposition has $O(n)$ nodes, it follows that there are at most $O(n2^{3\text{tw}(G)})$ recursive states. On the other hand, the worst case scenario for a computation within a recursive state occurs in join nodes, where the computing time is at most $O(\text{poly}(\text{tw}(G)) \cdot 2^{2\text{tw}(G)})$ using the representation of Lemma 5.1.2. Therefore, given the representation of Lemma 5.1.2 we need at most $O(\text{poly}(\text{tw}(G)) \cdot n2^{5\text{tw}(G)})$ time for computing $M[B, X, F]$. ■

Corollary 5.1.1. *The WNCS problem on bounded treewidth graphs is solvable in $O(\text{poly}(\text{tw}(G)) \cdot (n2^{5\text{tw}(G)} + n + m))$ time.*

Proof. First compute the representation of Lemma 5.1.2 in $O(\text{poly}(\text{tw}(G)) \cdot (n + m))$ time. Then compute $\min_{X \subseteq B} \{M[B, X, \emptyset]\}$, where B is the root of a nice tree decomposition of G . Note that this minimization does not add a multiplicative $O(2^{\text{tw}(G)})$ factor to the running time given in Theorem 5.1.2, because in the proof of Theorem 5.1.2 we counted for these possible recursive states. ■

5.2 An approximation scheme for planar graphs

We first introduce a slight variation of the WNCS problem which will be useful for obtaining a PTAS for WNCS on planar graphs. Given a vertex-weighted graph $G = (V, E)$, and a subset $E' \subseteq E$, we are to find a subset $S \subseteq V$ such that $E' \subseteq \bigcup_{v \in S} E(N[v])$. In other words, we are to cover the edges of E' using the vertices of G . The goal is to find S that minimizes $w(S)$. Within this subsection, we let (G, E') denote the instance of the problem.

Lemma 5.2.1. *There is a linear time algorithm for (G, E') if G is a bounded treewidth graph.*

Proof. Add a vertex v_e adjacent only to u and v for each edge $uv = e \in E \setminus E'$, and set $w_{v_e} = 0$. Let G' be the resulting graph. We claim that $\text{tw}(G') = \text{tw}(G)$ if $\text{tw}(G) \geq 2$; and $\text{tw}(G') = 2$ if $\text{tw}(G) = 1$. To see this, consider any tree decomposition of G such that every node has at most $\text{tw}(G) + 1$ vertices. For every $uv = e \in E \setminus E'$ add a neighboring node with vertices $\{v_e, u, v\}$ to some node that contains both, u and v . Clearly, the resulting tree is a tree decomposition of G' . Furthermore, $\text{tw}(G') = \text{tw}(G)$ if G had treewidth at least 2 and $\text{tw}(G') = 2$ otherwise.

Finally, if S is a solution for the problem (G, E') , then $S \cup \{v_e : e \in E \setminus E'\}$ is a NC-set for G' ; and if S' is a NC-set for G' , then $S' \cap V(G)$ is a solution for (G, E') . Therefore, it is enough to compute a minimum weight NC-set for G' , which is linear by Corollary 5.1.1. $\blacksquare$

Let $k \geq 4$. Assume G is a plane graph and its layers L_i ($i \in \mathbb{Z}$) defined as before. Suppose we fix some offset $l \in \{0, \dots, k-1\}$. For each $j \in \mathbb{Z}$, let $G_l^j = G[L_{l+kj} \cup \dots \cup L_{l+k(j+1)-1}]$. Observe that $\{V(G_l^j) : j \in \mathbb{Z}\}$ is a partition of vertices of G . Let $H_l^j = G[L_{l+kj-2} \cup \dots \cup L_{l+kj+1}]$. Observe that H_l^j contains two layers of G_l^{j-1} and two layers of G_l^j .

The idea is to optimally cover the “internal” edges for each G_l^j , and then “glue together” the solutions without incurring in a big error by carefully covering the remaining edges in H_l^j .

Solve the problems (G_l^j, T_l^j) for $j \in \mathbb{Z}$, where T_l^j are the edges that have both endpoints in the set $L_{l+kj+1} \cup \dots \cup L_{l+k(j+1)-2}$. Let S_l^j be an optimal solution of the problem (G_l^j, T_l^j) , and S_l be the set $\bigcup_j S_l^j$. Solve as well the problems (H_l^j, U_l^j) for $j \in \mathbb{Z}$, where U_l^j are the edges of H_l^j that have at least one endpoint in the set $L_{l+kj-1} \cup L_{l+kj}$. Let R_l^j be an optimal solution of the problem (H_l^j, U_l^j) , and R_l be the set $\bigcup_j R_l^j$. Let $O_l = R_l \cup S_l$. Observe that O_l is a NC-set for G for any l because $E = \bigcup_j T_l^j \cup U_l^j$. Our algorithm outputs $S = O_l$ that minimizes $w(O_l)$ over all the offsets $l \in \{0, \dots, k-1\}$.

Let S^* be an optimal weighted NC-set for G . In the following we prove that

$$w(S) \leq \left(1 + \frac{4}{k}\right) w(S^*).$$

Lemma 5.2.2. *There is an offset $l \in \{0, \dots, k-1\}$ such that*

$$w(R_l) \leq \frac{4}{k} w(S^*).$$

Proof. Let $B_l = \bigcup_j V(H_l^j)$. On the one hand, we have that $V(H_l^j) \cap V(H_l^{j+1}) = \emptyset$ because $k \geq 4$. Then, each layer of G is contained in exactly four different subgraphs H_l^j , and thus

$$\sum_{l=0}^{k-1} w(S^* \cap B_l) = 4w(S^*).$$

It follows that there is some $l \in \{0, \dots, k-1\}$ such that

$$w(S^* \cap B_l) \leq \frac{4}{k}w(S^*).$$

On the other hand, any edge of the set U_l^j can be covered only by vertices from $V(H_l^j)$ among the sets $V(H_l^{j'})$ ($j' \in \mathbb{Z}$). Therefore, $S^* \cap V(H_l^j)$ covers the edges in U_l^j . In other words, $w(R_l^j) \leq w(S^* \cap V(H_l^j))$. This implies

$$w(R_l) \leq w(S^* \cap B_l) \leq \frac{4}{k}w(S^*).$$

■

Let l be some offset as in the above Lemma. Since all the edges in T_l^j can only be covered by vertices from $V(G_l^j)$ among the sets $V(G_l^{j'})$, $j' \in \mathbb{Z}$, it follows that $w(S_l^j) \leq w(S^* \cap V(G_l^j))$, and thus $w(S_l) \leq w(S^*)$.

Observe that $w(S) = \min_{l'} w(O_{l'}) \leq w(O_l)$, and therefore $w(S) \leq (1 + \frac{4}{k}) w(S^*)$. We thus conclude the following result.

Theorem 5.2.1. *There is a PTAS for WNCS on planar graphs.*

Proof. For any $\epsilon > 0$, run the above algorithm with $k = \max \{4, \lceil \frac{4}{\epsilon} \rceil\}$. ■

5.3 Further remarks

Let $G = (V, E)$ be a graph with maximum degree $\Delta(G)$. We may model the WNCS problem as a set cover instance, where the universe to be covered is $E(G)$ and the sets are $\{e : v \text{ covers } e\} = E[v]$ for $v \in V(G)$. Observe that each edge $uv = e \in E$ can be covered by any vertex in the set $N[u] \cap N[v]$, and this set has cardinality at most $\Delta(G) + 1$. This means that each edge appears in at most $\Delta(G) + 1$ sets $E[v]$. By using the set cover approximation for bounded frequency systems given by Hochbaum [52] we obtain trivially an approximation ratio $\Delta(G) + 1$.

Question. Is it possible to improve the trivial $\Delta(G) + 1$ ratio for WNCS or NCS on bounded degree graphs?

On the other hand, it is known that deciding whether a graph has an NC-set of size at most k is NP-complete, even for line graphs with maximum degree 3 [47]. It would be interesting to find a constant approximation ratio for NCS or WNCS on line graphs. Recall that given a line graph L one can find some pre-image G of L in linear time due to an algorithm by Roussopoulos [79]. Therefore, a sensible approach would be to analyze the problem on the pre-image G . The interpretation in G of an NC-set is quite simple: we are to choose a set of edges in $E(G)$ such that the corresponding vertex set in $V(L)$ covers all the edges in L . There are many things one can observe when looking at the problem from this perspective. For example, given a vertex $v \in V(G)$ and two incident edges to v , e and f , the edge $ef \in E(L)$ can be covered only by edges of $E(G)$ that are incident to v (i.e., in $\delta(v)$) and the “opposite” edge to v that forms the triangle with edges e and f (if such edge exists). In other words, if there is some vertex v and a pair of edges $e, f \in \delta(v)$ such that they do not form a triangle (i.e. without an opposite edge), then any feasible solution must have at least one edge in $\delta(v)$. This suggests solving at first an edge-cover in G of the the set $\{v : \text{there are } e, f \in \delta(v) \text{ such that they do not form a triangle in } G\}$. The accurate way of doing this is modeling an edge cover on a graph where we add a zero weight edge incident to each v not in the described set. This may be computed in polynomial time (see, e.g., [80, Theorem 27.2]). It is easy to see that the cost of this edge cover is at most one time the optimum. However, there might remain uncovered edges of $E(L)$, but these have an interesting property. Say $ef \in E(L)$ is uncovered by the edge cover and v is the endpoint e and f share. By our modeling it follows that e and f are part of the same triangle in G . Moreover, since any edge in $\delta(v)$ would cover ef and ef is not covered, it must be the case that $N[v]$ induces

a complete subgraph in G . The standard problem arises. On the one hand, we need to establish a lower bound on some optimal cover of the remaining edges of $E(L)$. On the other hand, we need to compute some feasible cover and relate its size with that bound. A final comment: when covering some edge ef , it seems that there is no clear way of telling whether it is preferable to choose the opposite edge or an edge from $\delta(v)$ —instances that benefit either are easy to construct. Unfortunately, reasonable compromises as “choosing all three” or “choosing one in $\delta(v)$ and one opposite” seem to guarantee feasibility, but the bound becomes harder to establish. This is a problem we leave for further research.

Question. Is there a fixed constant approximation algorithm for NCS on line graphs?

Chapter 6

Neighborhood Independent Set

*“The pleasures arising from
thinking and learning will make
us think and learn all the more.”
—Aristotle
Nicomachean Ethics*

Let $G = (V, E)$ be a graph. A subset $S \subseteq E$ is *neighborhood independent (NI)* if $|E[v] \cap S| \leq 1$ for any vertex $v \in V$. The *maximum NI-set (NIS)* problem is to find a NI-set of maximum cardinality. The *weighted NI-set (WNIS)* problem is defined over edge-weighted graphs and the goal is to find a NI-set that maximizes its total weight.

In [Section 6.1](#) we argue that WNIS is hard to approximate. In [Section 6.2](#) we show that a simple greedy algorithm yields an approximation ratio Δ for general graphs. In [Section 6.3](#) we propose a fractional local ratio algorithm for diamond-free graphs with performance ratio $\Delta/2 + 1$. In [Section 6.4](#) we prove that WNIS is polynomial on cographs. We show a tight bound on the cardinality of a NI-set on regular graphs in [Section 6.5](#). In [Section 6.6](#) we show a linear time algorithm for weighted graphs with bounded treewidth. This last result is used in [Section 6.7](#) to obtain a PTAS on weighted planar graphs.

6.1 Inapproximability

Let us first briefly observe that the NI-set problem is hard to approximate within a ratio $O(\Delta^{1-\epsilon})$ for any $\epsilon > 0$. The implications of this are twofold: on the one hand, it rules out any constant approximation ratio; and on the other hand, it proves that our algorithms are in a sense tight.

Theorem 6.1.1. *For $\Delta \geq 4$, WNIS is NP-hard to approximate within a ratio*

$$\frac{\Delta - 1}{2^{O(\sqrt{\log(\Delta-1)})}}.$$

Proof. To show this we give a reduction from independent set that preserves approximability and use a hardness result given by Trevisan in [84].

Let G be a graph with $\Delta(G) \geq 3$ and suppose we are to compute an independent set of G . Construct the graph H as follows. For each vertex $u \in V(G)$ add two adjacent vertices u and u' to H . For each edge $uv \in E(G)$ add a new vertex c_{uv} in H adjacent to u, u', v and v' . For an edge of type uu' we set $w_{uu'} = 1$; and $w_e = 0$ for any edge e incident to some c_{uv} . There is a direct correspondence between NI-sets of H with no zero-weight edges and independent sets in G preserving their sizes: for an independent set I of G we define the NI-set $\{uu' : u \in I\}$ in H ; and for an NI-set S of H we define the independent set $\{u : uu' \in S\}$ in G . It is clear that a β -approximate NI-set of H amounts to a β -approximate independent set of G .

A key observation is that $\Delta(H) = \max\{\Delta(G) + 1, 4\} = \Delta(G) + 1$. Trevisan [84] observed that it is NP-hard to approximate the maximum independent set problem within a ratio $\Delta(G)/2^{O(\sqrt{\log \Delta(G)})}$. Therefore, it is NP-hard to approximate WNIS within a ratio

$$\frac{\Delta(H) - 1}{2^{O(\sqrt{\log(\Delta(H)-1)})}}.$$

■

Remark 4. *For any $\epsilon > 0$ and $\Delta \geq 4$, WNIS cannot be approximated within a ratio $O(\Delta^{1-\epsilon})$, unless $P=NP$.*

Proof. This follows because for any constant $c > 0$ we have

$$\Delta^{1-\epsilon} = \frac{\Delta}{2^{\epsilon \log \Delta}} = o\left(\frac{\Delta - 1}{2^{c\sqrt{\log(\Delta-1)}}}\right).$$

■

6.2 A Δ -approximation algorithm

If $\Delta \leq 2$, then computing a maximum weighted NI-set amounts to finding a maximum weighted matching; the set S is the maximum weighted matching. We thus consider graphs with $\Delta \geq 3$.

For an edge $uv = e \in E$, let $C(e) \subseteq E$ be the set of edges which are *in conflict* with e ; more formally,

$$C(e) = \bigcup_{w \in N[u] \cap N[v]} E[w].$$

Note that $e \in C(e)$.

We first introduce a technical result.

Lemma 6.2.1. *If S is a NI-set of G and $\Delta \geq 3$, then $|S \cap C(e)| \leq \Delta$ for each $uv = e \in E$.*

Proof. Let $W = N[u] \cap N[v] \setminus \{u, v\}$. Observe that $|W| \leq \Delta - 1$. If $|W| = \Delta - 1$, then $S \cap C(e) = S \cap \bigcup_{w \in W} E[w]$, and therefore

$$|S \cap C(e)| \leq \sum_{w \in W} |E[w] \cap S| \leq |W| = \Delta - 1.$$

If $|W| \leq \Delta - 2$, then $S \cap C(e) = S \cap (E[u] \cup E[v] \cup \bigcup_{w \in W} E[w])$, and therefore $|S \cap C(e)| \leq 2 + |W| \leq \Delta$. ■

Consider the natural greedy approach: begin with an empty solution S , and in each iteration add to S the edge $e \in E \setminus S$ of maximum weight such that $|(S \cup \{e\}) \cap E[v]| \leq 1$ for each $v \in V$. Return the constructed set once no more edges can be added.

Let S^* be an optimal NI-set of G . Each time we add an edge e to S , the edges in $C(e)$ cease to be candidates for future iterations—they become blocked. Among the edges in $C(e)$ some may have been blocked in a previous iteration while some in the current. Those in $S^* \cap C(e)$ blocked in a previous iteration are already accounted for by some $e' \in S$. And those in $S^* \cap C(e)$ blocked in the current iteration are at most Δ by the above lemma, and each of them has weight at most w_e by the choice of e . When the algorithm ends, all edges in S^* are blocked by S , and it follows that $w(S^*) \leq \Delta w(S)$. This implies an approximation ratio Δ .

Remark 5. *The analysis of the algorithm is tight.*

Proof. Consider the graph of Figure 6.2.1. The greedy algorithm outputs the sole edge of weight $1 + \epsilon$, while the optimum corresponds to the bold edges of total weight Δ . Therefore, the ratio between the optimum and the computed solution is $\Delta/(1 + \epsilon)$. ■

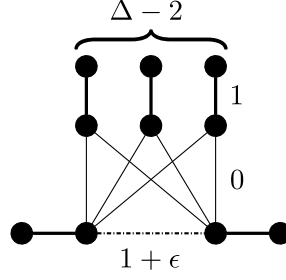


Figure 6.2.1: Tightness of the greedy algorithm.

6.3 An approximation algorithm for diamond-free graphs

In this section we present an algorithm using the fractional local ratio, which already appeared in the thesis when we explored induced matchings in [Section 2.1](#).

Let $G = (V, E)$ be an edge-weighted diamond-free graph with weights $w_e \in \mathbb{Q}_{\geq 0}$. Consider the following linear relaxation of the WNIS problem.

$$\begin{aligned} \max \quad & \sum_{e \in E} w_e x_e \\ \text{s.t.} \quad & x(E[v]) \leq 1 \quad \forall v \in V, \\ & x_e \geq 0 \quad \forall e \in E. \end{aligned}$$

For a subset $F \subseteq E$, we denote LP_F to the above linear program restricted to the variables x_f for $f \in F$.

In what follows we show that Algorithm 3 has performance ratio $\Delta/2+1$ provided that in each recursive call one can find an edge $e \in F$ such that $x(C(e)) \leq \Delta/2+1$. Note that there is at most one recursive call in each step, and in each iteration $|F|$ decreases by at least one. Therefore, the algorithm ends after at most $|E|$ recursive calls. The proof of the following theorem is the same as in Theorem 2.1.1; although for a different problem, ratio α and method of finding the decomposition of the weight vector.

Theorem 6.3.1. ([5, 20]). *Let $\alpha = \Delta/2 - 1$ and $F \subseteq E$. If for any non empty subset $F' \subseteq F$ and any feasible solution y to $LP_{F'}$ there is some $g \in F'$ such that $y(C(g)) \leq \alpha$, then Algorithm 3 computes a NI-set S such that $w(S) \geq \frac{1}{\alpha} \sum_{f \in F} w_f x_f$, where x is the solution to LP_F computed in Line 3.*

In what follows we show that there is always some edge $e \in F$ such that

Algorithm 3 NI(F, w)

```

1: if  $F = \emptyset$  then
2:   return  $\emptyset$ 
3: Compute an optimal solution  $x$  to  $LP_F$ 
4: Let  $F_0 = \{e \in F : x_e = 0\}$ 
5: if  $F_0 \neq \emptyset$  then
6:   return NI( $F \setminus F_0, w$ )
7: Let  $e \in F$  such that  $x(C(e)) \leq \Delta/2 + 1$ 
8: For each  $f \in F$ , let  $\widehat{w}_f = \begin{cases} w_e & \text{if } f \in C(e), \\ 0 & \text{otherwise.} \end{cases}$ 
9:  $S \leftarrow \text{NI}(F \setminus \{e\}, w - \widehat{w})$ 
10: if  $S \cup \{e\}$  is an NI-set (i.e. if  $C(e) \cap S = \emptyset$ ) then
11:    $S \leftarrow S \cup \{e\}$ 
12: return  $S$ 

```

$x(C(e)) \leq \Delta/2 + 1$. Note that since G is diamond-free, each edge is contained in exactly one clique.

Lemma 6.3.1. *Let x be an optimal solution to LP_F . If each clique K such that $x(E(K)) > 0$ has at least $\Delta/2 + 1$ vertices, then there is some clique K' such that $x(E(K')) \geq 1/2$.*

Proof. Let v be a vertex associated to a tight constraint of LP_F , that is, such that $x(E[v]) = 1$. Let K be a clique containing v . If $x(E(K)) \geq 1/2$, then there is nothing to prove. Suppose $x(E(K)) < 1/2$. Then there must be some other clique K' such that $x(E(K')) > 0$ containing v because of the tightness of v . Observe that $K \cap K' = \{v\}$ because K and K' cannot share another vertex since G is diamond-free. Since $\Delta \geq \deg(v) \geq |V(K)| + |V(K')| - 2 \geq \Delta$, it follows that we have equality throughout and the only cliques containing v are K and K' , and therefore $x(E(K')) > 1/2$. ■

Lemma 6.3.2. *Let x be an optimal solution to LP_F , then there is some edge $e \in F$ such that $x(C(e)) \leq \Delta/2 + 1$.*

Proof. First suppose that there is some clique K such that $x(E(K)) > 0$ and

$|V(K)| < \Delta/2 + 1$. Take an edge $e \in F \cap E(K)$. Clearly,

$$x(C(e)) \leq \sum_{w \in V(K)} x(E[w]) \leq |V(K)| < \frac{\Delta}{2} + 1.$$

If the above assumption does not hold, then we can suppose that for any clique K such that $x(E(K)) > 0$ we have $|V(K)| \geq \Delta/2 + 1$. By Lemma 6.3.1 there is some clique K such that $x(E(K)) \geq 1/2$. Take an edge $uv = e \in F \cap E(K)$. We have that

$$\begin{aligned} x(C(e)) &\leq x(E[u]) + x(E[v]) + \sum_{w \in V(K) \setminus \{u,v\}} x(E[w] \setminus E(K)) \\ &\leq 2 + \frac{|V(K)| - 2}{2} \\ &\leq 2 + \frac{\Delta - 2}{2} \\ &= \Delta/2 + 1, \end{aligned}$$

where the first inequality holds because each variable x_f with $f \in C(e)$ appears in the sum; the second because $x(E[w] \setminus E(K)) \leq 1 - x(E(K)) \leq 1/2$; and the third because if $|V(K)| = \Delta + 1$, then K must be an isolated clique, and therefore $x(C(e))$ would trivially be at most 1, which means that we can assume $|V(K)| \leq \Delta$. ■

Corollary 6.3.1. *Algorithm 3 has performance ratio $\Delta/2 + 1$.*

6.4 A polynomial time algorithm on cographs

Let us first restate the definition of cographs. Given two graphs G and H , the *join* graph $G \oplus H$ is the graph obtained after connecting all the vertices of G with all the vertices of H . A *cograph* is a graph that can be constructed using the following rules.

- $(\{v\}, \emptyset)$ is a cograph.
- If G and H are cographs, then $G \oplus H$ is a cograph.
- If G and H are cographs, then $G \cup H$ is a cograph.

A tree representing the above decomposition is called *cotree*, and it can be computed in linear time due to an algorithm by Corneil et al. [28].

In what follows we give the recursive rules for building a polynomial time algorithm using the cotree of G for computing a NI-set of maximum weight. Note that the unweighted case is linear for P_4 -tidy graphs [33, 87], which is a superclass of cographs. However, it is not clear how to generalize that algorithm for the weighted case.

The base and union cases are trivial. We are to analyze the $G \oplus H$ case. Let F denote the set of edges introduced when joining G and H .

Lemma 6.4.1. *If S is a NI-set of $G \oplus H$ and it contains an edge $e \in E(G) \cup E(H)$, then it is the only edge in S .*

Proof. Suppose $e \in E(G) \cap S$ and that there is another edge $f \in S$. If $f \in E(G)$, then for any vertex $v \in V(H)$ we have $|E[v] \cap S| \geq 2$. If $f \in E(H)$, then $|E[v] \cap S| \geq 2$ for each endpoint v of e and f . Finally, if $uv = f \in F$ with $v \in V(H)$, it follows that $|E[v] \cap S| \geq 2$ because v is adjacent to both endpoints of e . ■

The optimum of $G \oplus H$ will thus be either a sole edge from $E(G) \cup E(H)$ or some subset of F . In what follows we show how to find the best possible NI-set contained in F .

Let G' be the complete bipartite graph given by vertex sets $\{c_G(v) : v \in V(G)\}$ and $\{c_H(u) : u \in V(H)\}$, where $c_G(v)$ is the connected component of G containing v , and $c_H(u)$ the connected component of H containing u . Define the weight of an edge $c_G(v)c_H(u)$ as $\max\{w_{xy} : x \in c_G(v) \text{ and } y \in c_H(u)\}$.

Lemma 6.4.2. ([27]) *Each connected component of a cograph has diameter at most 2.*

Lemma 6.4.3. *If M is a maximum matching of G' and N a maximum weighted NI-set contained in F , then $w(M) = w(N)$.*

Proof. We first prove $w(M) \leq w(N)$. To see this it is enough to prove that M is a NI-set contained in F , where M in the edge set of $G \oplus H$ is given by the weight definition of G' . Under this interpretation it is clear that $M \subseteq F$. Suppose M is not a NI-set. Then, without losing overall generality, there must be some vertex $v \in V(G)$ such that $|E[v] \cap M| \geq 2$. Suppose $\{e, f\} \subseteq E[v] \cap M$. Note that if e has no endpoint in $c_G(v)$, then $e \notin E[v]$. In the same way, if f has no endpoint in $c_G(v)$, then $f \notin E[v]$. Therefore, e and f have both at least one endpoint in $c_G(v)$ —a contradiction because M is a matching.

We now prove $w(M) \geq w(N)$. First suppose that there are two edges $e, f \in N$ with endpoints in one connected component of G . Suppose these endpoints are u and v , respectively. Clearly, u and v must be different vertices. If $\text{dist}_G(u, v) = 1$, then $|E[v] \cap \{e, f\}| = |E[u] \cap \{e, f\}| = 2$. Finally, if $\text{dist}_G(u, v) = 2$, then for any vertex $w \in V(G)$ adjacent to u and v we have $|E[w] \cap \{e, f\}| = 2$. Note that by Lemma 6.4.2 it cannot be the case that $\text{dist}_G(u, v) > 2$. Therefore, each connected component of G and H contains at most one endpoint of N . In other words, N is a matching in G' . By the weight definition of G' , it follows that $w(M) \geq w(N)$. ■

Theorem 6.4.1. *The WNIS problem on cographs is polynomially solvable.*

Question. We ask whether WNIS can be solved in polynomial time for P_4 -tidy graphs.

6.5 A bound for regular graphs

In this section we present an asymptotically tight upper bound on the cardinality of a NI-set for d -regular graphs.

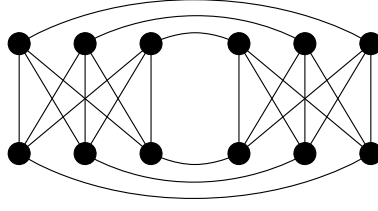
Theorem 6.5.1. *Let G be a d -regular graph and S a NI-set. Then $|S| < \frac{m}{d-1}$.*

Proof. First note that in a d -regular graph $2m = nd$. Suppose that $|S| \geq \frac{m}{d-1}$. Then, multiplying by 2 on each side we get $\sum_{v \in V} d_S(v) = 2|S| \geq \frac{2m}{d-1}$, where $d_S(v)$ is the number of edges in S incident to v . Therefore, by the pigeonhole principle, there must be some vertex $v \in V$ such that $d_S(v) \geq \frac{2m}{n(d-1)} = \frac{d}{d-1} > 1$, which is a contradiction since $|E[v] \cap S| \leq 1$. ■

Remark 6. *The bound is asymptotically tight. Indeed, for any $d \geq 2$ there is a d -regular graph G such that for a maximum NI-set S ,*

$$\frac{\frac{m}{d-1}}{|S|} = \frac{d}{d-1}.$$

Proof. Consider the graph given by two copies of $K_{(d-1), (d-1)}$ where each vertex is adjacent to its copy as in Figure 6.5.1. Clearly, $m = 2d(d-1)$ and $|S| = 2(d-1)$ because NI-sets are exactly matchings in triangle-free graphs. ■

Figure 6.5.1: Tightness of Theorem 6.5.1 for $d = 4$.

6.6 Weighted bounded treewidth graphs

Let $G = (V, E)$ be a graph. As in Chapter 5, we say that a vertex v *covers* an edge e if v is an endpoint of e or both endpoints of e are neighbors of v . Define

$$\text{cov}(X) = \{uv \in E : \exists x \in X \text{ where } u, v \in N[x]\} = \bigcup_{x \in X} E[x]$$

to be the edges of G covered by vertices in X . If U is a vertex set we let $E(U)$ denote the set of edges with both endpoints in U .

Suppose G has treewidth at most k and assume a nice tree decomposition T of G is given as input. We solve slightly more general version of the WNIS problem: given a node B , a subset $X \subseteq E(B)$, and a subset $W \subseteq B$, we are to find a subset $S \subseteq E_B$ of maximum weight such that

- (i) S is a NI-set of G_B ,
- (ii) $S \cap E(B) = X$, and
- (iii) $S \cap E_B \cap \text{cov}(W) \subseteq E(B)$.

We let $[B, X, W]$ denote an instance of this problem.

Intuitively, the vertices in W are “saturated” by edges picked up into the solution in the current or an ancestor state. We use this set for avoiding adding any edge that would saturate a previously saturated vertex (making thus the solution unfeasible). We need to keep W in a consistent state during the execution of the algorithm. Observe that it may be the case that we add an edge into the solution at a node B , but we cannot add to W all the vertices that are saturated by this edge (because they may not be in B). Therefore, in order to maintain feasibility we need to remember which edges we have added to the solution. For this purpose, we use the set X . The intuition behind condition (iii) is that S cannot pick any edge covered by W that it is not in the current node B and is lower in the tree decomposition.

In what follows we give the recursive rules for the dynamic program. We use a top-down approach with memoization. We let $M[B, X, W]$ denote the computed value for the instance $[B, X, W]$. For $e \in E(B)$, let $\xi_B(e) = \{w \in B : e \in E[w]\}$ be the set of saturated vertices by e in B . For convenience, given a subset $L \subseteq E$ we write $\xi_B(L) = \bigcup_{e \in L} \xi_B(e)$.

Suppose we are to solve $[B, X, W]$. We will assume $|E[u] \cap X| \leq 1$ for each $u \in B$ (i.e., the current state does not imply infeasibility), and $\xi_B(X) \subseteq W$ (that is, every saturated vertex by X from the set B is already listed in W). Depending upon the type of the node B , we have the following cases.

- *Leaf.* Set $M[B, X, W] = 0$.
- *Forget.* This is the only type of node where we can add new edges to the solution. Suppose B' is the child of B , $v \notin B$, and $B' = B \cup \{v\}$. If $|E[v] \cap X| \geq 2$, then the instance $[B', X, W]$ is unfeasible, and thus we set $M[B, X, W] = -\infty$. If $|E[v] \cap X| = 1$, then we cannot add any new edge of the set $\delta(v) \cap E(B')$ to the solution because v is saturated, and thus we set $M[B, X, W] = M[B', X, W \cup \{v\}]$. Finally, if $E[v] \cap X = \emptyset$, then we set $M[B, X, W] = \max\{M[B', X, W], M_v\}$, where

$$M_v = \max\{M[B', X \cup \{uv\}, W \cup \xi_{B'}(uv)] : uv \in \delta(v) \cap E(B') \setminus \text{cov}(W)\}.$$

Which can be thought of the best alternative between skipping an addition at this step, and the best solution we can build recursively if we add some previously unseen edge to the solution. Thus,

$$M[B, X, W] = \begin{cases} M[B', X, W \cup \{v\}] & \text{if } |E[v] \cap X| = 1, \\ \max\{M[B', X, W], M_v\} & \text{if } E[v] \cap X = \emptyset, \\ -\infty & \text{otherwise.} \end{cases}$$

- *Introduce.* Suppose $B = B' \cup \{v\}$, where B' is the child of B , and $v \notin B'$. We pay for the edges at the deepest nodes in the tree decomposition, which are always the introduce nodes. Furthermore, we need to remove the vertex v and update the sets X and W to be consistent with B' . We thus set $M[B, X, W] = M[B', X \setminus \delta(v), W \setminus \{v\}] + w(X \cap \delta(v))$.
- *Join.* Suppose B_1 and B_2 are the children of B . We need to avoid that some non-saturated vertex at state $[B, X, W]$ gets saturated by one edge in the

subgraph G_{B_1} and by other in G_{B_2} . In order to do so, we split the possible saturations between the two subtrees. Since we do not know the best splitting, we simply guess it. We thus compute

$$M[B, X, W] = \max_{T \subseteq B \setminus W} M[B_1, X, W \cup T] + M[B_2, X, W \cup \bar{T}] - w(X),$$

where $\bar{T} = (B \setminus W) \setminus T$.

Theorem 6.6.1. *Let $[B, X, W]$ be an instance such that $|E[u] \cap X| \leq 1$ for each $u \in B$ and $\xi_B(X) \subseteq W$. Then $M[B, X, W]$ is the value of an optimum solution for $[B, X, W]$ if there is a feasible solution, or $-\infty$ otherwise.*

Proof. We prove by induction on the tree decomposition T . In each case we will prove that $[B, X, W]$ is feasible if and only if some other instance is feasible, implying thus the proposition.

If B is a leaf of T , then there is nothing to prove. We have the following cases.

- *Forget.* Recall that $B' = B \cup \{v\}$ and $v \notin B$. Note that for this type of nodes $G_B = G_{B'}$, and therefore whenever we have a NI-set for one of them, it will be so for the other. We have the following cases.

- $|E[v] \cap X| = 1$. Suppose $[B, X, W]$ is feasible and let S^* be an optimal solution. We claim that S^* is feasible for $[B', X, W \cup \{v\}]$. Observe that $\delta(v) \cap S^* = \emptyset$, because otherwise there would be two edges in $E[v] \cap S^*$ (note that $X \subseteq E(B) = E(B') \setminus \delta(v)$).

(i) Trivial.

(ii) $S^* \cap E(B') = S^* \cap (E(B) \cup (\delta(v) \cap E(B'))) = S^* \cap E(B) = X$.

(iii) Since $X \subseteq S^*$, $|X \cap E[v]| = 1$ and $|S^* \cap E[v]| \leq 1$ (because S^* is a NI-set), it follows that $|S^* \cap E[v]| = 1$. Let $\{xy\} = S^* \cap E[v]$. We have that

$$\begin{aligned} S^* \cap E_{B'} \cap \text{cov}(W \cup \{v\}) &= (S^* \cap E_{B'} \cap \text{cov}(W)) \cup \\ &\quad (S^* \cap E_{B'} \cap E[v]) \\ &\subseteq E(B) \cup \{xy\} \\ &\subseteq E(B'). \end{aligned}$$

Thus, $w(S^*) \leq M[B', X, W \cup \{v\}]$.

Suppose now that $[B', X, W \cup \{v\}]$ is feasible and let S' be an optimal solution. Let us prove that S' is feasible for $[B, X, W]$.

- (i) Trivial.
- (ii) Since $X \subseteq E(B)$, $S' \cap E(B) = S' \cap E(B') \setminus \delta(v) = X \setminus \delta(v) = X$.
- (iii) $S' \cap E_B \cap \text{cov}(W) \subseteq S' \cap E_{B'} \cap \text{cov}(W \cup \{v\}) \subseteq S' \cap E(B') = X \subseteq E(B)$.

Since $[B, X, W]$ is feasible, let S^* be an optimal solution. It follows that $w(S^*) \geq w(S') = M[B', X, W \cup \{v\}]$. Combining both cases we conclude that $M[B', X, W \cup \{v\}]$ is the value of $[B, X, W]$.

- $E[v] \cap X = \emptyset$. Suppose $[B, X, W]$ is feasible and let S^* be an optimal solution. We claim that S^* is feasible for at least one subproblem considered in the maximization. We have the following two cases.

* $S^* \cap \delta(v) \cap E(B') \setminus \text{cov}(W) = \emptyset$. We are to show that S^* is feasible for $[B', X, W]$.

- (i) Trivial.
- (ii) We have

$$\begin{aligned} S^* \cap E(B') &= S^* \cap (E(B) \cup (\delta(v) \cap E(B'))) \\ &= (S^* \cap E(B)) \cup (S^* \cap \delta(v) \cap E(B')) \\ &= X \cup (S^* \cap \delta(v) \cap E(B')). \end{aligned}$$

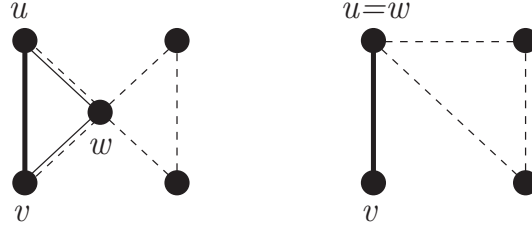
Suppose that $S^* \cap \delta(v) \cap E(B') \neq \emptyset$, then there is necessarily some $uv \in S^* \cap E(B')$. Since $S^* \cap \delta(v) \cap E(B') \setminus \text{cov}(W) = \emptyset$, it follows that $uv \in \text{cov}(W)$. We also know that $S^* \cap E_B \cap \text{cov}(W) \subseteq E(B)$. This implies that $uv \in E(B)$, but this is not possible because $v \notin B$. Therefore, $S^* \cap E(B') \cap \delta(v) = \emptyset$, and thus $S^* \cap E(B') = X$.

- (iii) $S^* \cap E_{B'} \cap \text{cov}(W) = S^* \cap E_B \cap \text{cov}(W) \subseteq E(B) \subseteq E(B')$.

Thus, S^* is feasible for $[B', X, W]$.

- * $S^* \cap \delta(v) \cap E(B') \setminus \text{cov}(W) \neq \emptyset$. In what follows we prove that S^* is feasible for some subproblem considered in M_v . Since S^* is a NI-set, we have that $S^* \cap \delta(v) \cap E(B') = \{uv\}$ for some $u \in B'$. Note that $uv \notin \text{cov}(W)$. We show that S^* is feasible for the instance $[B', X \cup \{uv\}, W \cup \xi_{B'}(uv)]$, which is considered in M_v because $uv \in \delta(v) \cap E(B') \setminus \text{cov}(W)$.

- (i) Trivial.

Figure 6.6.1: Possible configurations for uv and w . Any dotted edge may be e .

(ii)

$$\begin{aligned}
 S^* \cap E(B') &= S^* \cap (E(B) \cup (\delta(v) \cap E(B'))) \\
 &= (S^* \cap E(B)) \cup (S^* \cap \delta(v) \cap E(B')) \\
 &= X \cup \{uv\}.
 \end{aligned}$$

(iii) Since $\text{cov}(W \cup \xi_{B'}(uv)) = \text{cov}(W) \cup \text{cov}(\xi_{B'}(uv))$, it follows that

$$\begin{aligned}
 S^* \cap E_{B'} \cap \text{cov}(W \cup \xi_{B'}(uv)) &\subseteq E(B) \cup \\
 &\quad (S^* \cap E_{B'} \cap \text{cov}(\xi_{B'}(uv))).
 \end{aligned}$$

Clearly, $uv \in S^* \cap E_{B'} \cap \text{cov}(\xi_{B'}(uv))$. Suppose that there is some other edge $e \in S^* \cap E_{B'} \cap \text{cov}(\xi_{B'}(uv))$. Since $e \in \text{cov}(\xi_{B'}(uv))$, there is some $w \in \xi_{B'}(uv)$ such that $e \in E[w]$. There are two possibilities, either uvw is a triangle, or $u = w$ (or $v = w$). In either case, it follows that $|E[w] \cap S^*| \geq 2$ (for clarification see [Figure 6.6.1](#)), and thus S^* is not a NI-set for G_B . Therefore, it follows that

$$E(B) \cup (S^* \cap E_{B'} \cap \text{cov}(\xi_{B'}(uv))) \subseteq E(B) \cup \{uv\} \subseteq E(B').$$

Thus, S^* is feasible for the problem $[B', X \cup \{uv\}, W \cup \xi_{B'}(uv)]$.

The above two cases imply $w(S^*) \leq M[B, X, W]$. It remains to prove the other inequality. Suppose there is at least one feasible subproblem that appears in the expression of $M[B, X, W]$. Let S' be an optimal solution of such problem. There are two possible cases, in both of them we will show that S' is feasible for $[B, X, W]$.

* S' is a solution to $[B', X, W]$. It is clear that (i) holds. Since

$S' \cap E(B') = X$ and $X \subseteq E(B)$, it follows that $S' \cap E(B) = X$, and therefore (ii) holds. To see (iii), note that

$$S' \cap E_B \cap \text{cov}(W) = S' \cap E_{B'} \cap \text{cov}(W) \subseteq S' \cap E(B') = X \subseteq E(B).$$

* S' is a solution to some subproblem considered in M_v , say the instance $[B', X \cup \{uv\}, W \cup \xi_{B'}(uv)]$ for some $uv \in \delta(v) \cap E(B') \setminus \text{cov}(W)$.

(i) Trivial.

(ii) We know that $S' \cap E(B') = X \cup \{uv\}$. Since $E[v] \cap X = \emptyset$, it follows that $S' \cap E(B) = S' \cap E(B') \setminus \delta(v) = X$.

(iii)

$$\begin{aligned} S' \cap E_B \cap \text{cov}(W) &\subseteq S' \cap E_{B'} \cap \text{cov}(W \cup \xi_{B'}(uv)) \\ &\subseteq S' \cap E(B') \\ &= S' \cap (E(B) \cup (E(B') \cap \delta(v))) \\ &\subseteq E(B) \cup (S' \cap \delta(v)) \\ &= E(B) \cup \{uv\}. \end{aligned}$$

Since $uv \notin \text{cov}(W)$, it follows that $S' \cap E_B \cap \text{cov}(W) \subseteq E(B)$.

This implies that if S^* is an optimal solution for $[B, X, W]$, then

$$w(S^*) \geq w(S') = M[B, X, W].$$

– $|E[v] \cap X| \geq 2$. There is nothing to prove.

Summarizing, we have seen that if $[B, X, W]$ admits a feasible solution then there is at least one feasible subproblem in the right hand side, and that the converse is also true. It thus follows that the optimal value of $[B, X, W]$ is $M[B, X, W]$.

- *Introduce.* Suppose $[B, X, W]$ is feasible and let S^* be an optimal solution. We claim that $S^* \setminus \delta(v)$ is a feasible solution for $[B', X \setminus \delta(v), W \setminus \{v\}]$.

(i) $S^* \setminus \delta(v)$ is a NI-set for $G_{B'}$ because S^* is a NI-set for G_B and $G_{B'}$ is a subgraph of G_B .

(ii) $(S^* \setminus \delta(v)) \cap E(B') = (S^* \setminus \delta(v)) \cap (E(B) \setminus \delta(v)) = S^* \cap E(B) \setminus \delta(v) = X \setminus \delta(v)$.

(iii)

$$\begin{aligned}
(S^* \setminus \delta(v)) \cap E_{B'} \cap \text{cov}(W \setminus \{v\}) &\subseteq S^* \cap E_B \cap \text{cov}(W) \setminus \delta(v) \\
&\subseteq E(B) \setminus \delta(v) = E(B').
\end{aligned}$$

On the one hand, since $X \subseteq S^*$, it follows that $X \cap \delta(v) \subseteq S^* \cap \delta(v)$. On the other hand, we have $S^* \cap \delta(v) \subseteq S^* \cap E(B) = X$, because otherwise there would be an edge uv in $S^* \cap \delta(v)$ but not in $E(B)$, and thus uv must be in the subgraph $G_{B'}$, which does not contain the vertex v . Therefore, $S^* \cap \delta(v) \subseteq X \cap \delta(v)$, which implies that $S^* \cap \delta(v) = X \cap \delta(v)$. Combining these two observations we get

$$\begin{aligned}
w(S^*) &= w(S^* \setminus \delta(v)) + w(S^* \cap \delta(v)) \\
&\leq M[B', X \setminus \delta(v), W \setminus \{v\}] + w(X \cap \delta(v)).
\end{aligned}$$

Suppose now that $[B', X \setminus \delta(v), W \setminus \{v\}]$ is feasible and let S' be an optimal solution. In the following we prove that $\tilde{S} = S' \cup (X \cap \delta(v))$ is a feasible solution for $[B, X, W]$. Note that $\delta(v) \cap S' = \emptyset$.

(i) Let $w \in V_B$, we have the following two cases.

– $w = v$. We have that

$$E[v] \cap (S' \cup (X \cap \delta(v))) = (E[v] \cap S') \cup (E[v] \cap X \cap \delta(v)).$$

If there is some edge $e \in E[v] \cap S'$, then $e \in E(B')$ because T^B is a tree decomposition. It follows that $e \in E(B') \cap S' = X \setminus \delta(v)$. Therefore, $E[v] \cap S' \subseteq E[v] \cap X$. Moreover, $X \cap \delta(v) \subseteq X \cap E[v]$, and thus $E[v] \cap \tilde{S} \subseteq E[v] \cap X$. Since we have assumed that $|E[u] \cap X| \leq 1$ for each $u \in B$, it follows that $|E[v] \cap \tilde{S}| \leq 1$.

– $w \neq v$. We have that

$$E[w] \cap \tilde{S} = (E[w] \cap S') \cup (E[w] \cap X \cap \delta(v)).$$

Suppose there is some $e \in E[w] \cap X \cap \delta(v)$. Since $e \in \delta(v)$, $e \in E[w]$ and $v \notin V_{B'}$, it must be the case that $w \in B$. By assumption, we know that $\xi_B(e) \subseteq W$, and thus $w \in W \setminus \{v\}$. Recall that $S' \cap E_{B'} \cap \text{cov}(W \setminus \{v\}) \subseteq E(B')$ by the feasibility of S' . There-

fore, if there exists some $f \in E[w] \cap S'$, then $f \in E(B')$, and thus $f \in X$. This implies that $\{e, f\} \subseteq X$ and therefore $|E[w] \cap X| \geq 2$, which contradicts the assumption that for each $u \in B$ we have $|E[u] \cap X| \leq 1$. Therefore, it can not be the case that e and f exist as we described. Thus, if there is such e , then $E[w] \cap S' = \emptyset$, and thus $|E[w] \cap \tilde{S}| = 1$. If there is no such e , then $|E[w] \cap \tilde{S}| = |E[w] \cap S'| \leq 1$ by the feasibility of S' .

(ii)

$$\begin{aligned} E(B) \cap (S' \cup (X \cap \delta(v))) &= (E(B) \cap S') \cup (E(B) \cap X \cap \delta(v)) \\ &= (X \setminus \delta(v)) \cup (X \cap \delta(v)) \\ &= X. \end{aligned}$$

(iii) We are to prove that $\tilde{S} \cap E_B \cap \text{cov}(W) \subseteq E(B)$. First note that if there is some edge $xy \in S' \cap E_{B'} \cap E[v]$, then x and y must be in B because T^B is a tree decomposition, xyv is a triangle and $xy \in E_{B'}$. Therefore, $S' \cap E_{B'} \cap E[v] \subseteq E(B')$. Using this we get

$$\begin{aligned} \tilde{S} \cap E_B \cap \text{cov}(W) &= \tilde{S} \cap ((E(B) \cap \delta(v)) \cup E_{B'}) \cap \text{cov}(W) \\ &= ((\tilde{S} \cap E(B) \cap \delta(v)) \cup (\tilde{S} \cap E_{B'})) \cap \text{cov}(W) \\ &\subseteq E(B) \cup (S' \cap E_{B'} \cap \text{cov}(W)) \\ &\subseteq E(B) \cup (S' \cap E_{B'} \cap \text{cov}(W \setminus \{v\})) \cup \\ &\quad (S' \cap E_{B'} \cap E[v]) \\ &\subseteq E(B). \end{aligned}$$

Hence $\tilde{S}$ is a feasible solution for $[B, X, W]$. If S^* is an optimum solution for $[B, X, W]$, it follows that

$$\begin{aligned} w(S^*) &\geq w(\tilde{S}) \\ &= w(S') + w(X \cap \delta(v)) \\ &= M[B, X \setminus \delta(v), W \setminus \{v\}] + w(X \cap \delta(v)). \end{aligned}$$

And thus the value of the problem $[B, X, W]$ is $M[B, X, W]$.

- *Join.* Suppose $[B, X, W]$ is feasible and let S^* be an optimal solution. Let $S_1^* = S^* \cap E_{B_1}$ and $S_2^* = S^* \cap E_{B_2}$. In the following we prove that for

$T = \xi_B(S_2^*) \setminus W$, S_1^* and S_2^* are feasible for $[B_1, X, W \cup T]$ and $[B_2, X, W \cup \bar{T}]$, respectively.

- (i) Since S^* is a NI-set for G_B , $S_1^* \subseteq E_{B_1}$ and $S_2^* \subseteq E_{B_2}$, it follows that S_1^* and S_2^* are NI-sets for G_{B_1} and G_{B_2} , respectively.
- (ii) Since $E(B) = E(B_1) = E(B_2)$, then $S_1^* \cap E(B_1) = S_2^* \cap E(B_2) = X$.
- (iii) We prove this property for each subproblem separately.

– $[B_1, X, W \cup T]$.

$$\begin{aligned} S_1^* \cap E_{B_1} \cap \text{cov}(W \cup T) &= S_1^* \cap E_{B_1} \cap \text{cov}(W \cup (\xi_B(S_2^*) \setminus W)) \\ &= S_1^* \cap E_{B_1} \cap \text{cov}(W \cup \xi_B(S_2^*)) \\ &\subseteq E(B) \cup (S_1^* \cap E_{B_1} \cap \text{cov}(\xi_B(S_2^*))). \end{aligned}$$

Suppose that there is some $e \in S_1^* \cap E_{B_1} \cap \text{cov}(\xi_B(S_2^*))$ such that $e \notin E(B)$. Since $e \in \text{cov}(\xi_B(S_2^*))$, e is covered by some vertex $u \in B$, where u is saturated by some edge $e' \in S_2^* \subseteq E_{B_2}$. Observe that since $e \in E_{B_1} \setminus E(B)$ and $e' \in E_{B_2}$, it follows that $e \neq e'$. This implies $|E[u] \cap S^*| \geq 2$ —a contradiction. Therefore, $S_1^* \cap E_{B_1} \cap \text{cov}(\xi_B(S_2^*)) \subseteq E(B)$.

– $[B_2, X, W \cup \bar{T}]$. By definition we have that

$$\bar{T} = (B \setminus W) \setminus T = (B \setminus W) \setminus (\xi_B(S_2^*) \setminus W) = (B \setminus \xi_B(S_2^*)) \setminus W.$$

Hence,

$$\begin{aligned} S_2^* \cap E_{B_2} \cap \text{cov}(W \cup \bar{T}) &= S_2^* \cap E_{B_2} \cap \text{cov}(W \cup ((B \setminus \xi_B(S_2^*)) \setminus W)) \\ &= S_2^* \cap E_{B_2} \cap \text{cov}(W \cup (B \setminus \xi_B(S_2^*))) \\ &\subseteq E(B) \cup (S_2^* \cap E_{B_2} \cap \text{cov}(B \setminus \xi_B(S_2^*))). \end{aligned}$$

Suppose that there is some edge $e \in S_2^* \cap E_{B_2} \cap \text{cov}(B \setminus \xi_B(S_2^*))$ such that $e \notin E(B)$. Then there is some $u \in B \setminus \xi_B(S_2^*)$ such that $e \in E[u]$. By definition we have that $u \in \xi_B(e)$. Since $e \in S_2^*$, then $\xi_B(e) \subseteq \xi_B(S_2^*)$. This is a contradiction because $u \in B \setminus \xi_B(S_2^*)$ and $u \in \xi_B(S_2^*)$. Hence, $S_2^* \cap E_{B_2} \cap \text{cov}(B \setminus \xi_B(S_2^*)) \subseteq E(B)$.

Therefore, $w(S^*) = w(S_1^*) + w(S_2^*) - w(X) \leq M[B, X, W]$.

Conversely, suppose there is some $T \subseteq B \setminus W$ such that the subproblems $[B_1, X, W \cup T]$ and $[B_2, X, W \cup \bar{T}]$ are feasible. Take the set T that maximizes the expression $M[B, X, W]$. Let S_1 be an optimal solution for $[B_1, X, W \cup T]$, and S_2 for $[B_2, X, W \cup \bar{T}]$. In the following we show that $S = S_1 \cup S_2$ is feasible for $[B, X, W]$.

- (i) Suppose, for contradiction, that S it is not a NI-set for G_B . Then there must be at least two edges e_1 and e_2 from S that belong to $E[v]$ for some vertex $v \in V_B$. Note that $E_{B_1} \cap E_{B_2} = E(B)$, and thus $S_1 \cap S_2 = X$. It is clear that it cannot be that both, e_1 and e_2 , are at the same time in S_1 or S_2 . Therefore, $e_1 \in E_{B_1} \setminus E(B)$ and $e_2 \in E_{B_2} \setminus E(B)$. It follows that $v \in B$. We thus have $v \in W \cup T$ or $v \in W \cup \bar{T}$. Assume w.l.o.g. the former case, then it must be the case that $e_1 \in S_1 \cap E_{B_1} \cap \text{cov}(W \cup T)$, but by feasibility we have $S_1 \cap E_{B_1} \cap \text{cov}(W \cup T) \subseteq E(B_1) = E(B)$, which is a contradiction because $e_1 \notin E(B)$.
- (ii) Since $B = B_1 = B_2$, we have $S \cap E(B) = (S_1 \cap E(B)) \cup (S_2 \cap E(B)) = X$.
- (iii)

$$\begin{aligned}
 S \cap E_B \cap \text{cov}(W) &= (S_1 \cap E_B \cap \text{cov}(W)) \cup (S_2 \cap E_B \cap \text{cov}(W)) \\
 &\subseteq (S_1 \cap E_{B_1} \cap \text{cov}(W \cup T)) \cup (S_2 \cap E_{B_2} \cap \text{cov}(W \cup \bar{T})) \\
 &\subseteq E(B_1) \cup E(B_2) \\
 &= E(B).
 \end{aligned}$$

This implies that $[B, X, W]$ is feasible, and therefore for an optimal solution S^* we have $w(S^*) \geq w(S_1) + w(S_2) - w(X) = w(S)$.

Combining both cases we have that $M[B, X, W]$ is the value of an optimum solution to $[B, X, W]$.

■

Observe that for every considered state we have $W \subseteq B$ and $X \subseteq E(B)$. Hence there are at most $O(2^{\text{tw}(G)^3})$ possible states for each node B of T . Each recursive step can be computed in $O(1)$ time, and therefore the running time of this algorithm is $O(n)$.

Corollary 6.6.1. *The maximum weight neighborhood independent set is solvable in linear time for bounded treewidth graphs.*

Proof. Suppose B is the root node of a nice tree decomposition for G . Solve the maximization problem $M[B, X, \xi_B(X)]$ over $X \subseteq E(B)$ such that X is a NI-set for $G[B]$. ■

6.7 An approximation scheme for planar graphs

Assume a fixed integer $k \geq 2$. Let G be an edge-weighted plane graph with layers L_i ($i \in \mathbb{Z}$). For each offset $l \in \{0, \dots, k-1\}$ and $j \in \mathbb{Z}$ let

$$G_l^j = (V_l^j, E_l^j) = G[L_{l+kj} \cup \dots \cup L_{l+k(j+1)-1}].$$

Note that the number of layers in G_l^j is k (except maybe the border cases). For each G_l^j set the weights of the edges whose both endpoints are in L_{l+kj} or both in $L_{l+k(j+1)-1}$ to zero. Solve the weighted NI-set on each of these subgraphs using the algorithm described above. Let S_l^j be the edge set computed for G_l^j . We can assume that S_l^j contains no edge of weight zero. Let $S_l = \bigcup_j S_l^j$. We output the set $S = S_l$ that maximizes $w(S_l)$ over l . Since there is no edge in S_l^j that is covered by some vertex from $V_l^{j'}$ for some $j' \neq j$ and S_l^j is a NI-set for G_l^j , it follows that S_l is a NI-set for G for any offset l . In what follows we prove the approximation factor.

Let $F_l^j \subseteq E$ be the set of edges with both endpoints in $L_{l+kj-1} \cup L_{l+kj}$. Let $F_l = \bigcup_j F_l^j$ and S^* be an optimal NI-set for G . On the one hand,

$$\sum_{l=0}^{k-1} w(S^* \cap F_l) \leq 2w(S^*),$$

and therefore there is some $l \in \{0, \dots, k-1\}$ such that

$$w(S^* \cap F_l) \leq \frac{2}{k}w(S^*).$$

On the other hand, since $S^* \cap (E_l^j \setminus F_l)$ is a NI-set for G_l^j for each j , it follows that

$$w(S_l^j) \geq w(S^* \cap (E_l^j \setminus F_l)).$$

Therefore,

$$w(S_l) = \sum_{j \in \mathbb{Z}} w(S_l^j) \geq \sum_{j \in \mathbb{Z}} w(S^* \cap (E_l^j \setminus F_l)) = w(S^* \cap (E \setminus F_l)).$$

Finally, since $w(S^*) = w(S^* \cap (E \setminus F_l)) + w(S^* \cap F_l)$, we conclude

$$\left(1 - \frac{2}{k}\right) w(S^*) \leq w(S^*) - w(S^* \cap F_l) = w(S^* \cap (E \setminus F_l)) \leq w(S_l).$$

We therefore get the following result.

Theorem 6.7.1. *There is a PTAS for WNIS on planar graphs.*

Proof. For any $\epsilon > 0$, run the above algorithm with $k = \max \left\{ 2, \left\lceil \frac{2}{\epsilon} \right\rceil \right\}$. ■

Chapter 7

Conclusions and further work

*“Life is what you make of it.
You are dealt a pack of cards
... Your job is to make the best
of the cards that have been
handed out to you.”*

—Lee Kuan Yew

Guruswami and Rangan explored the complexity of the CTS, CIS, NCS, and NIS problems [47]. They proved their NP-completeness on planar graphs with maximum degree 3, and that CTS, CIS, and NIS are NP-complete when restricted to line graphs with bounded maximum degree. We explored the approximability of these four problems. We gave a PTAS for each of them in their weighted variations for planar graphs. Furthermore, we proposed approximation algorithms for CTS, CIS and NIS on line graphs. Thus, in a sense, part of this thesis is an answer to [47] from the approximability point of view.

On the other hand, we improved a long standing approximation ratio by Zito for weighted induced matchings and gave some simple graph classes where a greedy approach yields reasonable results.

It is of interest to characterize on a higher level the nature of the properties that were exploited for obtaining the algorithms in this thesis. The property used for obtaining polynomial time approximation schemes on planar graphs is quite evident: a planar representation enables one to divide the graph into big chunks of controlled size, such that in each of these chunks we can find a sub-solution that will outperform any optimal solution restricted to this chunk. Moreover, as we increase the size of the chunks, the outperforming sub-solutions get bigger and bigger, and thus the approximation ratio improves.

The greedy algorithms in this work exploit a property that is found in many packing problems: if the graph is sparse in some sense, then adding an element to the solution affects a small part of the graph—the key is to bound how small

this is and compare it with an optimum solution. Local ratio algorithms are in a sense similar, what differs is that the argument uses a fractional solution to build the bound instead of comparing the constructed solution with some hypothetical optimum. Thus, these two techniques rely heavily on the idea of locality.

In [Section 3.5](#) we gave an algorithm for a covering problem (CTS), using the idea that a big enough set of somehow related elements must span many structures that we are to cover. This is useful when it is not clear which addition of elements to the solution will enable us to make a good bound.

In what follows we list the main results of this thesis.

7.1 Results

- For clique transversal set.
 - A 3-approximation algorithm for unweighted line graphs.
 - A 4-approximation algorithm for weighted line graphs.
 - A practical 3-approximation algorithm for unweighted planar graphs.
 - A linear time algorithm for weighted bounded treewidth graphs.
 - A PTAS for weighted planar graphs.
 - A $\left\lceil \frac{\Delta(G)+1}{2} \right\rceil$ -approximation algorithm for unweighted graphs with $\Delta(G)$ bounded by a fixed constant.
- For clique independent set.
 - A linear time algorithm for weighted bounded treewidth graphs.
 - A PTAS for weighted planar graphs.
 - A $\max \left\{ 3, \frac{\Delta(G)+3}{5} + \epsilon \right\}$ -approximation algorithm for unweighted line graphs.
- For neighborhood covering set.
 - A linear time algorithm for weighted bounded treewidth graphs.
 - A PTAS for weighted planar graphs.
- For neighborhood independent set.
 - A $\Delta(G)$ -approximation algorithm for weighted graphs.
 - A $(\Delta(G)/2+1)$ -approximation algorithm for weighted diamond-free graphs.

- A polynomial time algorithm for weighted cographs.
- An asymptotically tight bound on the cardinality of a neighborhood independent set for regular graphs.
- A linear time algorithm for weighted bounded treewidth graphs.
- A PTAS for weighted planar graphs.
- For induced matching.
 - A $\Delta(G)$ -approximation algorithm for weighted graphs.
 - A $(2k - 3)$ -approximation algorithm for weighted $K_{1,k}$ -free graphs.
 - A k -approximation algorithm for weighted $(K_{1,k}, \text{chair})$ -free graphs.

7.2 Further work

The inapproximability results can be further explored for the problems considered in this thesis. Clearly, these questions are not of much interest for planar graphs since we gave a PTAS for most of them. For other graph classes, however, this question is an interesting and important one.

Besides the obvious proposal for searching improved approximation ratios, we point out the following questions regarding approximability.

- Can the trivial $\Delta(G) + 1$ ratio be improved for the WNCS problem? See [Section 5.3](#) for a brief discussion.
- Is there a constant approximation ratio for WNCS or NCS on line graphs? See [Section 5.3](#) as well.
- Is it possible to approximate the unweighted independent set problem within a constant ratio for triangle-free graphs? In [Section 4.1](#) we relate this problem with the clique independence number.
- Is there a constant approximation algorithm for the neighborhood independent set on line graphs?
- In [\[33, 87\]](#) the authors solved NIS in linear time for P_4 -tidy graphs. In [Section 6.4](#) we gave a polynomial time algorithm for WNIS on cographs—a subclass of P_4 -tidy graphs. We ask whether WNIS can be solved in polynomial time for P_4 -tidy graphs.

Bibliography

- [1] Jochen Alber and Rolf Niedermeier. Improved tree decomposition based algorithms for domination-like problems. In *LATIN 2002: Theoretical Informatics, 5th Latin American Symposium, Cancun, Mexico, April 3-6, 2002, Proceedings*, volume 2286 of *Lecture Notes in Computer Science*, pages 613–628. Springer, 2002.
- [2] Javier Ignacio Arregui. Algoritmos eficientes para calcular el clique transversal mínimo en grafos, Tesis de Licenciatura. 2011.
- [3] Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *The Journal of the ACM*, 41(1):153–180, January 1994.
- [4] V. Balachandran, P. Nagavamsi, and C. Pandu Rangan. Clique transversal and clique independence on comparability graphs. *Information Processing Letters*, 58(4):181 – 184, 1996.
- [5] Reuven Bar-Yehuda, Keren Bendel, Ari Freund, and Dror Rawitz. Local ratio: A unified framework for approximation algorithms - In memoriam: Shimon Even 1935-2004. *ACM Comput. Surv.*, 36(4):422–463, 2004.
- [6] Piotr Berman and Toshihiro Fujito. On the approximation properties of independent set problem in degree 3 graphs. In *Algorithms and Data Structures, 4th International Workshop, WADS '95, Kingston, Ontario, Canada, August 16-18, 1995, Proceedings*, volume 955 of *Lecture Notes in Computer Science*, pages 449–460. Springer, 1995.
- [7] Piotr Berman and Martin Fürer. Approximating maximum independent set in bounded degree graphs. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '94*, pages 365–371, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.
- [8] Alan A. Bertossi. Dominating sets for split and bipartite graphs. *Information Processing Letters*, 19(1):37 – 40, 1984.
- [9] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the Twenty-fifth Annual ACM Symposium*

- on Theory of Computing*, STOC '93, pages 226–234, New York, NY, USA, 1993. ACM.
- [10] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11(1-2):1–21, 1993.
- [11] Hans L. Bodlaender. A partial k -arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.
- [12] Hans L. Bodlaender. Discovering treewidth. In *SOFSEM 2005: Theory and Practice of Computer Science, 31st Conference on Current Trends in Theory and Practice of Computer Science, Liptovský Ján, Slovakia, January 22-28, 2005, Proceedings*, volume 3381 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.
- [13] Hans L. Bodlaender. Treewidth: Characterizations, applications, and computations. In *Graph-Theoretic Concepts in Computer Science, 32nd International Workshop, WG 2006, Bergen, Norway, June 22-24, 2006, Revised Papers*, volume 4271 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.
- [14] Hans L. Bodlaender and Rolf H. Möhring. The pathwidth and treewidth of cographs. *SIAM Journal on Discrete Mathematics*, 6(2):181–188, 1993.
- [15] Andreas Brandstädt and Raffaele Mosca. On distance-3 matchings and induced matchings. *Discrete Applied Mathematics*, 159(7):509 – 520, 2011.
- [16] Kathie Cameron. Induced matchings. *Discrete Applied Mathematics*, 24(1):97 – 102, 1989.
- [17] Kathie Cameron. Induced matchings in intersection graphs. *Discrete Mathematics*, 278(1–3):1 – 9, 2004.
- [18] Kathie Cameron, R. Sritharan, and Yingwen Tang. Finding a maximum induced matching in weakly chordal graphs. *Discrete Mathematics*, 266(1–3):133 – 142, 2003.
- [19] Kathie Cameron and Tracy Walker. The graphs with maximum induced matching and maximum matching the same size. *Discrete Mathematics*, 299(1–3):49 – 55, 2005.

-
- [20] Stefan Canzar, Khaled M. Elbassioni, Gunnar W. Klau, and Julián Mestre. On Tree-Constrained Matchings and Generalizations. *Algorithmica*, 71(1):98–119, 2015.
 - [21] Gerard J. Chang, Martin Farber, and Zsolt Tuza. Algorithmic aspects of neighborhood numbers. *SIAM Journal on Discrete Mathematics*, 6(1):24–29, 1993.
 - [22] Jou-Ming Chang. Induced matchings in asteroidal triple-free graphs. *Discrete Applied Mathematics*, 132(1–3):67 – 78, 2003.
 - [23] Maw-Shang Chang, Ton Kloks, and Chuan-Min Lee. Maximum clique transversals. In *Graph-Theoretic Concepts in Computer Science, 27th International Workshop, WG 2001, Boltenhagen, Germany, June 14-16, 2001, Proceedings*, volume 2204 of *Lecture Notes in Computer Science*, pages 32–43. Springer, 2001.
 - [24] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.
 - [25] Vašek Chvatal. *Linear Programming*. Series of books in the mathematical sciences. W. H. Freeman, 1983.
 - [26] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.
 - [27] Derek G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3(3):163–174, 1981.
 - [28] Derek G. Corneil, Yehoshua Perl, and Lorna K. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985.
 - [29] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12 – 75, 1990.
 - [30] Konrad K. Dabrowski, Marc Demange, and Vadim V. Lozin. New results on maximum induced matchings in bipartite graphs and beyond. *Theoretical Computer Science*, 478:33 – 40, 2013.
 - [31] Márcio A. Duarte, Felix Joos, Lucia D. Penso, Dieter Rautenbach, and Uéverton S. Souza. Maximum induced matchings close to maximum matchings. *Theoretical Computer Science*, 588:131 – 137, 2015.

-
- [32] William Duckworth, David F. Manlove, and Michele Zito. On the approximability of the maximum induced matching problem. *Journal of Discrete Algorithms*, 3(1):79 – 91, 2005.
 - [33] Guillermo Durán, Martín Safe, and Xavier Warnes. Neighborhood covering and independence on P_4 -tidy graphs and tree-cographs. *Annals of Operations Research*, Nov 2017.
 - [34] Guillermo Durán, Min Chih Lin, Sergio Mera, and Jayme L. Szwarcfiter. Algorithms for finding clique-transversals of graphs. *Annals of Operations Research*, 157(1):37–45, 2008.
 - [35] Guillermo Durán, Min Chih Lin, and Jayme L. Szwarcfiter. On clique-transversals and clique-independent sets. *Annals of Operations Research*, 116(1-4):71–77, 2002.
 - [36] Paul Erdős, Tibor Gallai, and Zsolt Tuza. Covering the cliques of a graph with vertices. *Discrete Mathematics*, 108(1–3):279 – 289, 1992.
 - [37] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS*, 87:47–77, 2005.
 - [38] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.
 - [39] Gerd Fricke and Renu Laskar. Strong matchings on trees. *Congr. Numer.*, 89:239–239, 1992.
 - [40] Takuro Fukunaga and Hiroshi Nagamochi. Generalizing the induced matching by edge capacity constraints. *Discrete Optimization*, 4(2):198 – 205, 2007.
 - [41] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
 - [42] Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics*. Springer Publishing Company, Incorporated, 2nd edition, 2010.
 - [43] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 2004.

-
- [44] Martin Charles Golumbic and Renu C. Laskar. Irredundancy in circular arc graphs. *Discrete Applied Mathematics*, 44(1):79 – 89, 1993.
 - [45] Martin Charles Golumbic and Moshe Lewenstein. New results on induced matchings. *Discrete Applied Mathematics*, 101(1–3):157 – 165, 2000.
 - [46] Zvi Gotthilf and Moshe Lewenstein. Tighter approximations for maximum induced matchings in regular graphs. In *Approximation and Online Algorithms, Third International Workshop, WAOA 2005, Palma de Mallorca, Spain, October 6-7, 2005, Revised Papers*, volume 3879 of *Lecture Notes in Computer Science*, pages 270–281. Springer, 2005.
 - [47] Venkatesan Guruswami and C. Pandu Rangan. Algorithmic aspects of clique-transversal and clique-independent sets. *Discrete Applied Mathematics*, 100(3):183–202, 2000.
 - [48] András Gyárfás, Dieter Kratsch, Jenő Lehel, and Frederic Maffray. Minimal non-neighborhood-perfect graphs. *Journal of Graph Theory*, 21(1):55–66, 1996.
 - [49] Magnús M. Halldórsson. Approximations of independent sets in graphs. In *Approximation Algorithms for Combinatorial Optimization, International Workshop APPROX’98, Aalborg, Denmark, July 18-19, 1998, Proceedings*, volume 1444 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1998.
 - [50] Magnús M. Halldórsson and Hoong Chuin Lau. Low-degree graph partitioning via local search with applications to constraint satisfaction, max cut, and coloring. *Journal of Graph Algorithms and Applications*, 1(3):1–13, 1997.
 - [51] Michael A. Henning and Dieter Rautenbach. Induced matchings in subcubic graphs without short cycles. *Discrete Mathematics*, 315–316:165 – 172, 2014.
 - [52] Dorit S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982.
 - [53] Shiow-Fen Hwang and Gerard J. Chang. k -neighborhood-covering and -independence problems for chordal graphs. *SIAM Journal on Discrete Mathematics*, 11(4):633–643, 1998.
 - [54] Felix Joos. Induced matchings in graphs of bounded maximum degree. *SIAM Journal on Discrete Mathematics*, 30(3):1876–1882, 2016.

-
- [55] Felix Joos and Viet Hang Nguyen. Induced matchings in graphs of degree at most 4. *SIAM Journal on Discrete Mathematics*, 30(1):154–165, 2016.
 - [56] Felix Joos, Dieter Rautenbach, and Thomas Sasse. Induced matchings in subcubic graphs. *SIAM Journal on Discrete Mathematics*, 28(1):468–473, 2014.
 - [57] Ross J. Kang, Matthias Mnich, and Tobias Müller. Induced matchings in subcubic planar graphs. *SIAM Journal on Discrete Mathematics*, 26(3):1383–1411, 2012.
 - [58] Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.
 - [59] C. W. Ko and F. B. Shepherd. Adding an identity to a totally unimodular matrix. *Working paper LSEOR.94.14 London School of Economics, Operational Research Group 1994*.
 - [60] Daniel Kobler and Udi Rotics. Finding maximum induced matchings in subclasses of claw-free and P_5 -free graphs, and in graphs with matching and induced matching of equal maximum size. *Algorithmica*, 37(4):327–346, 2003.
 - [61] Arie M. C. A. Koster, Hans L. Bodlaender, and Stan P. M. van Hoesel. Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics*, 8:54–57, 2001.
 - [62] Chandra Mohan Krishnamurthy and R. Sritharan. Maximum induced matching problem on hhd-free graphs. *Discrete Applied Mathematics*, 160(3):224 – 230, 2012.
 - [63] Casimir Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15(1):271–283, 1930.
 - [64] Chuan-Min Lee. Weighted maximum-clique transversal sets of graphs. *ISRN Discrete Mathematics*, 2011.
 - [65] Chuan-Min Lee and Maw-Shang Chang. Distance-hereditary graphs are clique-perfect. *Discrete Applied Mathematics*, 154(3):525 – 536, 2006.
 - [66] Jenö Lehel. Neighbourhood-perfect line graphs. *Graphs and Combinatorics*, 10(2):353–361, 1994.

-
- [67] Jenő Lehel and Zsolt Tuza. Neighborhood perfect graphs. *Discrete Mathematics*, 61(1):93 – 101, 1986.
- [68] Zuosong Liang and Erfang Shan. Approximation algorithms for clique-transversal sets and clique-independent sets in cubic graphs. *Information Processing Letters*, 111(23-24):1104–1107, 2011.
- [69] Zuosong Liang and Erfang Shan. The clique-transversal set problem in claw-free graphs with degree at most 4. *Information Processing Letters*, 115(2):331 – 335, 2015.
- [70] Zuosong Liang, Erfang Shan, and Liying Kang. The clique-transversal set problem in $\{\text{claw}, K_4\}$ -free planar graphs. *Information Processing Letters*, 118:64 – 68, 2017.
- [71] Daniel Lokshantov, Martin Vatshelle, and Yngve Villanger. Independent set in P_5 -free graphs in polynomial time. In *SODA*, pages 570–581. SIAM, 2014.
- [72] Vadim V. Lozin. On maximum induced matchings in bipartite graphs. *Information Processing Letters*, 81(1):7 – 11, 2002.
- [73] Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *FOCS*, pages 17–27. IEEE Computer Society, 1980.
- [74] Hannes Moser and Somnath Sikdar. The parameterized complexity of the induced matching problem. *Discrete Applied Mathematics*, 157(4):715 – 727, 2009.
- [75] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.
- [76] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*. Wiley interscience series in discrete mathematics and optimization. Wiley, 1988.
- [77] Yury Orlovich, Gerd Finke, Valery Gordon, and Igor Zverovich. Approximability results for the maximum and minimum maximal induced matching problems. *Discrete Optimization*, 5(3):584 – 593, 2008.

-
- [78] Dieter Rautenbach. Two greedy consequences for maximum induced matchings. *Theoretical Computer Science*, 602:32 – 38, 2015.
 - [79] Nicholas D. Roussopoulos. A $\max\{m, n\}$ algorithm for determining the graph H from its line graph G . *Information Processing Letters*, 2(4):108 – 112, 1973.
 - [80] Alexander Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.
 - [81] Larry J. Stockmeyer and Vijay V. Vazirani. NP-completeness of some generalizations of the maximum matching problem. *Information Processing Letters*, 15(1):14 – 19, 1982.
 - [82] A. Takaoka, S. Tayu, and S. Ueno. Weighted dominating sets and induced matchings in orthogonal ray graphs. In *Proceedings of CoDIT*, pages 69–73, 2014.
 - [83] El-Ghazali Talbi. *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009.
 - [84] Luca Trevisan. Non-approximability results for optimization problems on bounded degree instances. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 453–461. ACM, 2001.
 - [85] Zsolt Tuza. Covering all cliques of a graph. *Discrete Mathematics*, 86(1–3):117 – 126, 1990.
 - [86] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.
 - [87] Xavier Warnes. Structural and algorithmic results on neighborhood-perfect graphs and neighborhood numbers. Master’s thesis, Departamento de Matemática, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Argentina, 2014.
 - [88] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.
 - [89] Gerhard J. Woeginger. *Exact Algorithms for NP-Hard Problems: A Survey*, pages 185–207. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

- [90] J. Wu. Neighborhood-covering and neighborhood-independence in strongly chordal graphs. *preprint*.
- [91] Michele Zito. Induced matchings in regular graphs and trees. In *Graph-Theoretic Concepts in Computer Science, 25th International Workshop, WG'99, Ascona, Switzerland, June 17-19, 1999, Proceedings*, volume 1665 of *Lecture Notes in Computer Science*, pages 89–100. Springer, 1999.