EXACT AND HEURISTIC APPROACHES FOR THE EDGE CLIQUE COVER PROBLEM

Douglas Picciani de Souza

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Abilio Pereira de Lucena Filho

Rio de Janeiro
Outubro de 2025

EXACT AND HEURISTIC APPROACHES FOR THE EDGE CLIQUE COVER PROBLEM

Douglas Picciani de Souza

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientador: Abilio Pereira de Lucena Filho

Aprovada por: Prof. Nelson Maculan Filho
Prof. Pedro Henrique González Silva
Prof. Luiz Satoru Ochi
Prof. Yuri Abitbol de Menezes Frota

RIO DE JANEIRO, RJ – BRASIL
OUTUBRO DE 2025

Picciani de Souza, Douglas

Exact and Heuristic Approaches for the Edge Clique Cover Problem/Douglas Picciani de Souza. – Rio de Janeiro: UFRJ/COPPE, 2025.

XVI, 119 p.: il.; 29, 7cm.

Orientador: Abilio Pereira de Lucena Filho

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2025.

Referências Bibliográficas: p. 84 – 87.

1. Edge Clique Cover. 2. Branch-and-Price. 3. Clique Sampling. I. Pereira de Lucena Filho, Abilio. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Para minha amada mãe,
Glorinha.

Acknowledgements

May the reader excuse me on this page, but I take the liberty to speak freely in my mother tongue — Afinal, esta é a página de agradecimentos, que exige de mim uma expressividade tal que só o português me permite entregar.

Foram oito anos de trabalho, prolongados pela pandemia e por problemas pessoais, e muitas pessoas fizeram parte da minha trajetória. Entretanto, dado o caráter bilateral do caminho que este trabalho percorreu, não há quem seja mais importante que o prof. Abilio, o meu orientador: A ti eu deixo o meu mais sincero muito obrigado. Afinal, após inúmeras reuniões semanais, eu não somente aprendi o ofício da pesquisa em otimização combinatória, mas também aprendi, da maneira mais improvável, sobre a força de ser um homem plenamente presente nas vidas daqueles que se ama. Obrigado por tudo que você me proporcionou e por sempre lembrar que sou capaz.

Agradeço aos integrantes da minha banca de defesa do doutorado, professores Nelson Maculan, Yuri Abtibol, Luiz Satoru e Pedro Henrique, pelo tempo aqui empregado em avaliar e sugerir pontos de melhora para o texto aqui exposto, muito obrigado.

Aqui também deixo um agradecimento aos meus pais, que dentro de seus limites fizeram o melhor que podiam para construir os pilares que suportam a pessoa que me tornei. Em especial à minha mãe, que sempre me lembrou de me manter firme e terminar o que comecei, seja lá qual dificuldade for.

Agradeço à minha irmã Natasha e ao meu cunhado Alexander, que nesses últimos anos, por pura sorte minha, vieram fazer parte da minha rotina e, consequentemente pela dualidade desse feliz reencontro, me trouxeram para mais perto da nossa família. Sem vocês, tudo teria sido mais complicado nesse trabalho. Muito obrigado.

Agradeço à Maeve, minha única sobrinha, a quem amo infinitamente. Tive o imenso prazer da sua presença nestes seus primeiros 3 anos de vida, me permitindo viver um mundo de constante novidade e observação. Sua presença é o sequestro certeiro dos pensamentos que prateiam barba e cabelo. Recentemente enfiou duas rodelas de azeitonas nos meus olhos, clamou às coleguinhas de sala pela individualidade da nossa relação dizendo "Esse é o MEU tio Dodo!", descobriu a coragem para andar no escuro ao colar um selo na testa escrito "bolso especial para roupas mol-

hadas" e acabou de vir aqui colocar uma lasca de chocolate na minha boca enquanto escrevo. Obrigado por estar aqui comigo.

Agradeço à Débora, cuja presença há mais de 25 anos flerta, indubitavelmente, com seu desejo em me ver inteiro em todas as esferas possíveis da minha vida. Parte do homem que sou, forjado fui pela força de tua palavra e de tua vida, às quais sou profundamente grato e privilegiado. É em nossa amizade que parte do traçado da vida se fez, ao me permitir espreitá-la por janelas que sozinho eu jamais abriria.

Agradeço ao meu padrinho Nicolas, que sempre esteve ao meu lado celebrando as minhas vitórias e me amparando nas dificuldades. Sou profundamente grato pela proximidade, pela mão sempre estendida e pelo exemplo de força e bondade.

Agradeço à Geysa, pela qual tenho a mais alta estima. Uma vez questionou-me quem, para mim, ela era. Entretanto, pelas vicissitudes da vida, eu não soube expressar o ímpeto que eu escondia. Aqui deixo essas breves palavras a ti, pessoa central, que foste, além da mais crua inspiração quanto à gana pela excelência, a faísca para que da minha primeira morte, eu, como homem, enxergasse plenamente a vida. Meu sincero muito obrigado.

Agradeço aos amigos que fiz durante o curso da matemática aplicada (UFRJ) e na ENSTA, que se fazem presentes mesmo que de longe: Cláudio Verdun, Guilherme Sales, Carlos Lechner, Daniel Soares, Paulo Ricardo, Mohammed. Deixo aqui agradecimentos especiais ao Cláudio, que em momentos-chave me proporcionou ajuda e caminhos que foram imensamente importantes na minha formação; e ao Guilherme, que sempre fazia contato naturalmente, para a minha sorte, nos momentos de maior solidão. Eu sou uma pessoa extremamente afortunada em ter vocês na minha vida.

Agradeço aos amigos recentes que ganhei em Araruama, que estiveram comigo, que cuidam da minha família, que torceram pelo meu sucesso e me deram o apoio para manter a minha sanidade nos últimos tempos: Layana, Roberta, Beatriz, Luís, Otávio e Gisele (a.k.a. minha melhor amiga).

Agradeço imensamente ao Fábio Ramos, que, em boa parte do meu doutorado, foi pessoa-chave ao me proporcionar oportunidades de trabalho que permitiram a continuidade e conclusão do meu doutorado. Muito obrigado por, diversas vezes, depositar uma confiança no meu labor que, por vezes, me faltava. Sem você, minha vida teria sido completamente diferente, e mais penosa, durante todos esses anos.

Agradeço ao grupo de pesquisa do Laboratório de Computação de Alto Desempenho e Aprendizado de Máquina em Engenharia (L'CADAME) do qual faço parte atualmente: Hamid, Daniel, Reza, Farhad, Gustavo, entre outros, que além de me incluir em calorosas confraternizações, me proporcionou acesso crucial à infraestrutura computacional de HPCs para a realização de parte das experimentações presentes nesta tese.

Agradeço ao Programa de Engenharia de Sistemas e Computação (PESC) pela oportunidade que me foi cedida e pela maneira como fui tratado durante todos esses anos. Em especial um muito obrigado ao prof. Luigi Simonetti, que, assim como o mestre dos magos, aparecia na minha vida, me aconselhava sobre partes cruciais para a implementação dos meus algoritmos, mudando completamente algumas das rotas que eu seguia, e simplesmente sumia.

Por fim, agradeço à UFRJ, que apesar das diversas lutas, foi casa minha durante os últimos 17 anos.

Muito obrigado a todos que, direta ou indiretamente, foram importantes nessa caminhada e, acima de tudo, obrigado por serem quem são.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

EXACT AND HEURISTIC APPROACHES FOR THE EDGE CLIQUE COVER PROBLEM

Douglas Picciani de Souza

October/2025

Advisor: Abilio Pereira de Lucena Filho

Department: Systems Engineering and Computer Science

The Edge Clique Cover Problem (ECCP) seeks to find the minimum number of cliques that cover all edges in a given graph. Despite its fundamental importance in combinatorial optimization and many applications such as computational geometry and network analysis, etc., ECCP remains computationally challenging due to its NP-hard nature and its resistance to polynomial-time approximation. This thesis presents a comprehensive investigation of Set Covering Problem formulations for ECCP, developing both advanced heuristic frameworks and novel exact solution methods. Part I introduces conceptual frameworks for algorithm design through systematic clique sampling, developing the Clique Sampling Heuristic (CSH), Reduced-Cost Sampling Heuristic (RCSH), and Local Search enhanced RCSH (LS-RCSH) that generate high-quality restricted formulations and significantly outperform traditional ECCP heuristics. Part II addresses fundamental limitations preventing robust exact algorithms by introducing a branch-and-price framework based on systematic graph transformation from ECCP to equivalent Vertex Clique Cover Problems (VCCP), enabling adaptation of branching strategies, from the Graph Coloring Problem, while preserving Maximum Weight Clique Problem structure throughout enumeration trees. Additionally, we develop a novel representative-based VCCP formulation that serves as both a standalone method and benchmark for the branch-and-price approach. Extensive computational experiments across diverse instance classes reveal distinct performance regimes where smaller and sparser instances remain tractable for exact methods while larger and denser graphs present significant challenges. The heuristic frameworks demonstrate robust performance across the entire test spectrum, with LS-RCSH achieving a reasonable balance between solution quality and efficiency, while the exact branch-and-price method successfully provides

meaningful bounds on some instances where direct SCP formulations fail. The contributions extend beyond specific algorithms to include methodological insights into clique-based optimization, systematic SCP formulation techniques, and principled branching strategy adaptation that establish foundations for future research in this challenging combinatorial optimization domain.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

Douglas Picciani de Souza

Outubro/2025

Orientador: Abilio Pereira de Lucena Filho

Programa: Engenharia de Sistemas e Computação

O Problema de Cobertura de Arestas por Cliques (ECCP) busca encontrar o número mínimo de cliques que cobrem todas as arestas de um grafo dado. Apesar de sua importância fundamental em otimização combinatória e em muitas aplicações tais como geometria computacional, análise de redes, etc., o ECCP permanece computacionalmente desafiador devido à sua natureza NP-difícil e à sua resistência ao desenvolvimento de algoritmos aproximativos em tempo polinomial. Esta tese apresenta uma investigação abrangente de formulações do Problema de Cobertura de Conjuntos (SCP) para o ECCP, desenvolvendo tanto estruturas heurísticas avançadas quanto métodos exatos de solução inovadores. A Parte I introduz estruturas conceituais para projeto de algoritmos através de amostragem sistemática de cliques, desenvolvendo a Heurística de Amostragem de Cliques (CSH), a Heurística de Amostragem por Custo Reduzido (RCSH), e a RCSH aprimorada com Busca Local (LS-RCSH) que geram formulações restritas de alta qualidade e superam significativamente heurísticas tradicionais para ECCP. A Parte II aborda limitações fundamentais que impedem algoritmos exatos robustos introduzindo uma estrutura branch-and-price baseada em transformação sistemática de grafos do ECCP para Problemas de Cobertura de Vértices por Cliques (VCCP) equivalentes, permitindo adaptação de estratégias de ramificação, proveniente do Problema de Coloração de Grafos, enquanto preserva a estrutura do Problema de Clique de Peso Máximo ao longo das árvores de enumeração. Adicionalmente, desenvolvemos uma formulação VCCP novel baseada em representantes com estratégias de redução sistemática que serve tanto como método independente quanto benchmark para a abordagem branch-and-price. Experimentos computacionais extensivos através de diversas classes de instâncias revelam regimes de desempenho distintos onde instâncias menores e mais esparsas permanecem tratáveis para métodos exatos enquanto grafos maiores e mais densos apresentam desafios computacionais significativos. As

estruturas heurísticas demonstram desempenho robusto através de todo o espectro de testes, com LS-RCSH alcançando equilíbrio entre qualidade de solução e eficiência, enquanto o método branch-and-price exato fornece com sucesso limites significativos em algumas instâncias onde formulações SCP diretas falham. As contribuições estendem-se além de desenvolvimentos algorítmicos específicos para incluir insights metodológicos em problemas de otimização baseados em cliques, técnicas sistemáticas de formulação SCP, e adaptação principiada de estratégias de ramificação que estabelecem fundações para pesquisas futuras neste domínio desafiador da otimização combinatória.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Edge Clique Cover Problem (ECCP) represents one of the fundamental challenges in combinatorial optimization, requiring the identification of the minimum number of cliques necessary to cover all edges in a given graph. Despite its seemingly straightforward formulation, ECCP exhibits remarkable computational complexity, being both NP-hard (ORLIN, 1977; KOU *et al.*, 1978) and resistant to polynomial-time approximation algorithms (LUND e YANNAKAKIS, 1994). This inherent difficulty, combined with the problem's diverse applications spanning computational geometry (AGARWAL *et al.*, 1994), compiler optimization (RAJAGOPALAN *et al.*, 2000), complex network analysis (CONTE *et al.*, 2016), and bioinformatics (BLANCHETTE *et al.*, 2012), has motivated decades of algorithmic research aimed at developing both practical heuristic approaches and theoretically rigorous combinatorial solution methods.

The computational intractability of ECCP stems from its fundamental relationship to the structure of graph cliques, complete subgraphs that represent the densest possible connectivity patterns within networks. While small instances can be solved through exhaustive enumeration of all maximal cliques, this approach quickly becomes computationally prohibitive as graph size and density increase. The exponential growth in the number of maximal cliques, particularly in dense graphs, creates a formidable barrier to scalable exact algorithms and necessitates sophisticated algorithmic frameworks that can navigate the combinatorial explosion while maintaining solution quality.

Current approaches to ECCP can be broadly categorized into two complementary research directions. The first focuses on heuristic methods that sacrifice optimality guarantees in favor of computational tractability, enabling the treatment of large-scale instances within reasonable time constraints. These approaches typically employ greedy strategies, local search techniques, or sampling-based methods to construct feasible solutions without exhaustively exploring the solution space. While such methods prove invaluable for practical applications involving massive

networks, they provide limited theoretical insights into problem structure and offer no guarantees regarding solution quality relative to the optimal value.

The second research direction pursues exact algorithms that guarantee optimal solutions through systematic exploration of the complete solution space. Traditional exact approaches for ECCP have relied primarily on specialized search tree algorithms with sophisticated data reduction techniques (GRAMM *et al.*, 2006; HEVIA *et al.*, 2023), achieving remarkable success on very sparse instances but struggling with moderate-to-high density graphs. The development of more robust exact methods requires addressing fundamental algorithmic challenges, including the design of effective branching strategies, the integration of powerful bounding techniques, and the management of exponentially large solution spaces.

This thesis contributes to both research directions through a comprehensive investigation of Set Covering Problem (SCP) formulations for ECCP, where the problem is reformulated as covering all graph edges using a minimum number of maximal cliques. This reformulation perspective enables the application of well-established techniques from integer programming and column generation theory (DESROSIERS e LÜBBECKE, 2011; LÜBBECKE e DESROSIERS, 2005) while revealing new structural properties that can be exploited algorithmically. The SCP formulation framework provides a unifying lens through which both heuristic and exact approaches can be systematically developed, analyzed, and compared.

The research presented in this thesis is organized into two distinct yet complementary parts that collectively address the spectrum of algorithmic approaches for ECCP. Part I focuses on conceptual frameworks for algorithm design through systematic clique sampling and restricted formulations, building upon our published work (PICCIANI e LUCENA, 2025). This investigation demonstrates how clever sampling strategies, including random selection, linear programming reduced cost criteria, and local search enhancement, can generate computationally tractable restricted formulations while maintaining high solution quality. Through the development of techniques such as the Clique Sampling Heuristic (CSH), the Reduced-Cost Sampling Heuristic (RCSH), and the enhanced Local Search Reduced-Cost Sampling Heuristic (LS-RCSH), this part establishes that column generation approaches combined with sophisticated sampling can yield high-quality solutions for challenging instances where traditional heuristics (CONTE *et al.*, 2016) fail.

Part II explores alternative approaches to exact ECCP algorithms by investigating a branch-and-price framework based on graph transformation. The approach converts ECCP instances into Vertex Clique Cover Problems (VCCP) through graph transformation and develops direct solution methods for VCCP formulations, representing an unexplored research direction in the current literature.

The branch-and-price framework is complemented by the development of a novel

VCCP formulation based on representatives, which provides an alternative mathematical perspective that enriches the comparative analysis while emphasizing the inherent computational complexity of the problem. This representative-based formulation, combined with systematic reduction strategies and cutting plane generation techniques, enables comprehensive benchmarking of the proposed exact methods across diverse instance classes.

The experimental evaluation encompasses systematic computational studies designed to assess both the practical performance and theoretical properties of the proposed algorithms. For the heuristic frameworks in Part I, the evaluation focuses on solution quality, computational efficiency, and scalability characteristics across varying graph structures and densities. The exact methods in Part II are evaluated through rigorous comparison with the natural baseline approach of solving the complete SCP formulation using all maximal cliques, emphasizing both optimality achievement and computational tractability boundaries.

The contributions of this thesis extend beyond the specific algorithms developed to include methodological insights into the design of effective optimization approaches for clique-based problems. The systematic investigation of SCP formulation techniques, the integration of column generation with local search, and the principled adaptation of branching strategies from related problems provide a foundation for future algorithmic development in this challenging domain. Furthermore, the comprehensive experimental analysis reveals important characteristics of problem difficulty across different instance classes, providing guidance for practitioners and researchers working with clique cover problems in various application contexts.

The remainder of this thesis is structured to present these contributions in a logical progression that builds from foundational concepts to sophisticated exact methods, culminating in experimental validation and analysis of all proposed approaches.

Part I

Conceptual Clique Sampling Frameworks for ECCP

Chapter 2

Introduction to Part I

Let $G = (V, E)$ be an undirected graph with a set of vertices V and a set of edges E. Denote by $E(C)$ the subset of edges of E with both end vertices in $C \subseteq V$ and by $G[C] = (C, E(C))$ the subgraph of G induced by C. A *c*lique is a nonempty set of vertices $C \subseteq V$ for which $|C| \geq 4$ applies and $G[C]$ is a complete graph. For convenience, we will extend this definition to also include complete induced subgraphs $G[C]$ with $|C| \in \{2, 3\}$. In doing so, an *Edge Clique Cover* (ECC) (JOHNSON e GAREY, 1979) is defined as a covering of all edges of G with subgraphs induced by cliques. The smallest cardinality possible for an ECC is denoted by κ and is known as the *ECC number* of G. Finally, the *ECC Problem* (ECCP) asks for an ECC of G with cardinality κ.

In the literature, ECCP is also known as the *R-Content Covering Problem* ORLIN (1977), the *Keyword Conflict Problem* KOU *et al.* (1978), the *Covering by Cliques Problem* GAVRIL (1972) and the *Intersection Graph Basis Problem* JOHNSON e GAREY (1979). It originates from some fundamental Set Theory questions posed by BOOLE (1952) and finds applications in widely scattered areas such as Computational Geometry AGARWAL *et al.* (1994), Compiler Optimization RAJAGOPALAN *et al.* (2000), Applied Statistics PIEPHO (2004); GRAMM *et al.* (2007), Artificial Intelligence MARKHAM e GROSSE-WENTRUP (2020), Complex Networks CONTE *et al.* (2016); GUILLAUME e LATAPY (2004) and Biology BLANCHETTE *et al.* (2012). Finally, the ECC number of a graph is also called its *Intersection Graph* (IG) number ROBERTS (1985); ERDÖS *et al.* (1966); MCKEE e MCMORRIS (1999). Accordingly, any reference to the IG number of a graph also qualifies as a reference to ECCP.

ECCP is known to be *NP*-hard, as proved by ORLIN (1977) and KOU *et al.* (1978). Furthermore, as indicated by LUND e YANNAKAKIS (1994), ECCP is hard to approximate efficiently. In more detail, the authors proved that there exists an $\epsilon > 0$ such that ECCP is not approximable within a factor of $|V|^{\epsilon}$, unless $P = NP$. Additionally, in tune with this result, HERMELIN *et al.* (2012) show that ECCP is

hard to approximate even for biconnected graphs of degree 7.

As far as exact solution ECCP procedures are concerned, two algorithms are currently available in the literature, GRAMM *et al.* (2006) and HEVIA *et al.* (2023). Both of them are of the *Search Tree* type and are very much dependent on data reductions implemented over the input graph $G = (V, E)$ (see GRAMM *et al.* (2006); HEVIA *et al.* (2023) for details on these reductions). The algorithm in HEVIA *et al.* (2023), on top of implementing the reduction procedures suggested in GRAMM *et al.* (2006) also benefits from some additional ones, suggested in HEVIA *et al.* (2023), and appears to have an edge over its counterpart.

In order to perform empirically well, the algorithm in HEVIA *et al.* (2023) requires input graphs that are amenable to the reduction tests it implements. Candidate graphs are typically required to be very sparse. However, this condition alone does not ensure a good empirical performance for it. The authors tested their procedure on: (a) 70 Erdős-Rényi graphs with $|V|$ ranging from 64 to 2048 and densities d ranging from 1.4% to 10% and (b) 36 very large real world graphs of extremely low densities. As an example, the smallest graph in (b) involves $5,242$ vertices and $14,484$ edges, which amounts to $d = 0.0105440470\%$. The largest one, on the other hand, involves $1,965,206$ vertices, $2,766,607$ edges and $d = 0.0000143270\%$. Relying on a computer with 1.5 terabytes of RAM and using a single thread, the algorithm managed to solve all Erdős-Rényi instances extremely fast (45 of them by reduction tests alone), with two single exceptions: namely, one corresponding to a graph with $|V| = 64$ and $d = 2\%$, the other to a $|V| = 64$ and $d = 3.7\%$. For solving these instances 2,646 and 457 CPU seconds were respectively required. Conversely, for their real world graphs, a CPU time limit of 24 hours was imposed on any run of the algorithm and 27 out of the 36 test instances were solved to proven optimality: 14 in under 1 CPU second, 6 in between 1 and 10 CPU seconds, 2 in respectively 20 and 36 CPU seconds, two in respectively 569 and 982 seconds and the final 3 in respectively 6,754, 10,728 and 12,967 CPU seconds. Furthermore, 16 out of these 27 instances were solved by reduction tests alone. The algorithm in HEVIA *et al.* (2023) thus qualifies as a very valuable contribution to an important niche application of ECCP.

Given the ECCP hardness to approximation, if optimality requirements are dropped, heuristics then become the option of choice for solving the problem. Recently, CONTE *et al.* (2016) and ABDULLAH e HOSSAIN (2022) introduced fast ECCP heuristics, geared into applications involving massive (complex network) graphs. The heuristic in CONTE *et al.* (2016) is characterized by a running time that is almost linear in $|E|$ and an $O(|V| + |E|)$ memory space requirement. The algorithm also possesses some additional attractive features. For instance, it is easy to adapt to objective functions other than the standard one, which asks for the

minimum number of ECCP cliques. Furthermore, based on our own computational experience with it, in addition to *running fast*, the algorithm is also competitive (in terms of solution quality) with the other existing ECCP heuristics in the literature, i.e., RAJAGOPALAN *et al.* (2000); GRAMM *et al.* (2006); PIEPHO (2004). Computational experiments in HEVIA *et al.* (2023), comparing the heuristics in CONTE *et al.* (2016) and ABDULLAH e HOSSAIN (2022), resulted in a larger number of best solutions being attained by the latter heuristic. Nevertheless, they also show that the heuristic in ABDULLAH e HOSSAIN (2022) appears less robust than the one in CONTE *et al.* (2016): namely, for some instances in the test bed it attains much worse results than its counterpart while the reverse never occurs. All previous considerations taken into account, we then opted for using the heuristic in CONTE *et al.* (2016), within some of the ECCP heuristics we propose here and also for comparison purposes.

The investigations in this thesis are centered on ECCP algorithms, exact or heuristic, based on the Set Covering Problem (SCP) BORNDÖFER (1998) formulation of the problem. More precisely, we focus on attempting to identify relevant aspects that should be taken into account in the design of new algorithms for solving this type of ECCP formulation.

The SCP formulation of ECCP is defined by the set of all distinct maximal cliques of $G = (V, E)$, our input graph. We denote it the SCP-ECCP formulation of ECCP and stress that it is aimed at finding a covering of the edges of E with the edges of a minimum number of subgraphs induced by maximal cliques of G. A maximal clique, one should note, is a clique that is contained in no larger clique.

Our option for the SCP-ECCP formulation is motivated by the fact that similar formulations for problems that are conceptually close to ECCP, such as the Vertex Coloring Problem (VCP) MEHROTRA e TRICK (1996), for instance, typically attain Linear Programming Relaxation (LPR) bounds that are *relatively close* to optimal solution values. Depending on the type of algorithm one has in mind, the massive number of cliques involved might be dealt with in various different ways. For example, in an exact ECCP Branch-and-Price (BP) algorithm they would be dealt with implicitly, by resorting to a Pricing Problem (PP) that generates cliques only as they become necessary (see DESROSIERS e LÜBBECKE (2011), for BP and PP details). On the other hand, when the focus is on the design of ECCP heuristics, one might restrict oneself to a small set of *attractive* maximal cliques of G. These cliques would then give rise to a Restricted SCP-ECCP (R-SCP-ECCP) formulation which might fail to contain an optimal solution to ECCP. The driving idea here, however, is to aim for an R-SCP-ECCP formulation with an optimal solution value that is hopefully close to κ.

In order to identify attractive maximal cliques for an R-SCP-ECCP formulation,

one might resort, for instance, to a PP based implicit column generation scheme. Alternatively, one might make multiple calls to a fast ECCP heuristic, such as CONTE *et al.* (2016), under different initialization conditions. This may be understood as an *explicit* column generation scheme. Either way, once the R-SCP-ECCP formulation is established one would then solve it, exactly or heuristically, and thus obtain a feasible solution to ECCP.

To establish the R-SCP-ECCP formulations investigated in this thesis, we resorted to implicit and explicit column generation schemes. Furthermore, in addition to solving these formulations and thus heuristically obtaining solutions to ECCP, we also explore a somewhat novel approach where we combine PP based column generation with *Local Search* (LS) STÜTZLE (1998). In this case, LS is carried out around a *good quality* feasible ECCP solution, obtained from the R-SCP-ECCP formulation in hand. Furthermore, when used recursively, LS tends to generate a sequence of improving ECCP solutions. Finally, the approach is quite general and is easily adapted to other problems solvable via PP based column generation algorithms.

To have an idea of how close our heuristic solutions come to κ, we initially generate SCP-ECCP formulations for all instances in our test set and attempt to solve them to proven optimality. To do so, we use the Bron-Kerbosch algorithm (BKA), as refined in TOMITA *et al.* (2006), to enumerate all maximal cliques of G and thus have access to the formulations. Next, we submit them to the Mixed Integer Programming (MIP) solver Gurobi, version 11. Rounded up LPR values for these formulations define a lower bound on κ and thus provide us with a baseline value for evaluating the quality of our heuristic solutions. Furthermore, κ values, when accessible to us, i.e., when Gurobi manages to solve the SCP-ECCP formulations to proven optimality, are clearly the best baseline values one may have. As one shall see, we only managed to obtain them for our less challenging to solve test instances. However, as one will also see, upon reaching our imposed run time limit, Gurobi runs frequently return feasible ECCP solutions that improve upon those obtained by procedures that rely on the heuristic in CONTE *et al.* (2016).

Apart from this introduction, the Part I of this thesis contains five additional chapters. Chapter 3 addresses the generation of baseline values for evaluating our ECCP heuristics. Next, in Chapter 4, we focus on experiments to obtain and solve R-SCP-ECCP formulations that are based on the heuristic in CONTE *et al.* (2016). Still focused on the generation of R-SCP-ECCP formulations, Chapter 5 investigates the use of PP based column generation schemes. Next, the results obtained in Chapter 5 are then attempted to be improved, in Chapter 6, with our proposed LS procedure. Finally, Part I is closed in Chapter 7, with some conclusions for the transition to Part II of this thesis. For better readability, a Table of Acronyms

is provided in Appendix B, while Appendix C organizes the data results for each Chapter of Part I into detailed tables.

Chapter 3

Baseline Values for Evaluating ECCP Heuristics

A very straightforward Integer Programming (IP) approach to ECCP is to generate and solve the SCP-ECCP formulation to the problem. In more detail, a formulation that associates a *column* (variable) to every distinct maximal clique of the input graph $G = (V, E)$ and a *row* to every distinct edge of E. For our empirical investigations, the set $\mathcal{C} = \{C_j \subset V : j \in J\}$, $J = \{1, ..., n\}$, of all maximal cliques of G is generated by the BKA variant suggested in TOMITA *et al.* (2006), as implemented by us. For every clique $C_j \in \mathcal{C}$ thus obtained, the edges of G which are *covered* by C_j are those in the subgraph of G induced by C_j, i.e., the edges $E(C_j)$ of subgraph $G[C_j] = (C_j, E(C_j))$.

Resorting to BKA to generate all maximal cliques of G imposes some obvious computational limitations. Among others, the most obvious ones are those associated with processing time and computer memory requirements. However, our main intent here is to have direct access to as good as possible lower and upper bounds on κ, for every instance in our test bed, no matter the CPU time required. In our particular case, we managed to generate all maximal cliques in $\mathcal{C}$ for any of our test graphs. As a result, SCP-ECCP formulations for any of these instances are thus available to us. However, as one shall see, the amount of RAM at our disposal is not sufficient to upload some of them into Gurobi.

With the exception of SCP-ECCP formulations for two of our test graphs, we managed to generate LPR bounds for all remaining ones. Better still, in some cases, we also obtained proven optimal solutions for them. As previously indicated, we relied on the Mixed Integer Programming (MIP) solver Gurobi, version 11, to generate LPR bounds and also to eventually find proven optimal solutions for our SCP-ECCP formulations.

3.1 The SCP-ECCP formulation to ECCP

Given a graph $G = (V, E)$, associate a variable $x_j \in [0, 1]$ with every maximal clique $C_j \in \mathcal{C}$. Additionally, identify by $J_e \subseteq J$ those cliques (variables) of $\mathcal{C}$ that *cover* edge $e \in E$, i.e., cliques that simultaneously contain both end vertices of e. Finally, consider a polyhedral region $\mathcal{R}_1$, defined as

$$\sum_{j \in J_e} x_j \geq 1, e \in E \tag{3.1}$$

$$x_j \geq 0, j \in J. \tag{3.2}$$

The SCP-ECCP formulation to ECCP is then given by

$$\text{minimize} \left\{ \sum_{j \in J} x_j : \mathbf{x} \in \mathcal{R}_1 \cap \mathbb{Z}^n \right\}, \tag{3.3}$$

and its corresponding LPR is defined as

$$\text{minimize} \left\{ \sum_{j \in J} x_j : \mathbf{x} \in \mathcal{R}_1 \right\}, \tag{3.4}$$

where variables $\mathbf{x}$ need not be further restrict with constraints

$$x_j \leq 1, j \in J, \tag{3.5}$$

as we explain next. In more detail, note that (3.3) is a minimization problem with nonnegative objective function coefficients, only. Accordingly, the feasibility enforced by $x_j = 1$, $j \in J$, is no different from the one enforced (at a higher cost) by $x_j \in \mathbb{N}$, $x_j \geq 2$. Hence, (3.5) is naturally implied by the objective function in (3.4).

Let us now attest the validity of (3.3). For any input graph $G = (V, E)$ and $e \in E$, recall, from the definition of $\mathcal{C}$, that there must exist at least one maximal clique $C_j \subseteq \mathcal{C}$ such that $G[C_j]$ contains e. In conjunction with this, note that inequalities (3.1) enforce that every edge of E must be covered by at least one of these cliques (variables). Finally, observe that the objective function in (3.3) asks for a minimum number of cliques to cover all edges of E.

3.2 The CGM heuristic

We denote by CGM the ECCP heuristic introduced in CONTE *et al.* (2016) and will describe its main features in the sequel. CGM is initialized with an input graph $G = (V, E)$. Next, it randomly selects an yet uncovered edge $\{u, v\} \in E$, thus

defining a clique C of size 2. This clique is then progressively expanded, one vertex a time, up to a point where it becomes a maximal clique of G. Throughout this process all edges entering $G[C]$ are labelled "covered". Next, the procedure iterates, provided any edge of E remains to be covered. As before, this is done by randomly selecting an yet uncovered edge to initialize a new clique, to be expanded into a maximal one. CGM comes to an end when no more uncovered edges of G exist. Algorithm 1 is a pseudo-code for CGM while Algorithm 2 is a pseudo-code for the function *findACliqueOf*, which is part of CGM.

Algorithm 1 *CGM*

Require: $G = (V, E)$
Ensure: An ECC *ecc* for G
1: Mark all edges in E as *uncovered*
2: $ecc \leftarrow \emptyset$
3: **while** there are uncovered edges **do**
4: $(u, v) \leftarrow$ SelectAnUncoveredEdge()
5: $C \leftarrow$ FindACliqueOf(u, v)
6: $ecc \leftarrow ecc \cup C$
7: Mark all edges of C as covered
8: **end while**
9: **return** *ecc*

In order to give a flavour on how function *findACliqueOf* operates, we will firstly indicate the main structures it relies on: namely, *open* and *uncovered open neighborhoods* for the vertices of V. We denote by $N(u) = \{v : \{u, v\} \in E\}$ the open neighborhood of vertex $u \in V$. Additionally, we denote by $U \subseteq E$ the set of yet uncovered edges of E and by $N_U(u) = \{v : \{u, v\} \in U\}$ the uncovered open neighborhood of u, defined by all vertices $v \in N(u)$ for which $\{u, v\} \in E$ is still uncovered. Function *findACliqueOf* takes as an input a yet uncovered randomly selected edge $\{u, v\} \in E$ that initializes a vertex set $R \subseteq V$ with its end vertices, u and v. Out of this initial clique of size 2, the function then attempts to enlarge R into a maximal clique that not only covers edge $\{u, v\}$ but, hopefully, will also cover as many yet uncovered edges of E as possible. This goal is attempted in a greedy fashion by the *extract_node* function, which selects the next vertex to enter R. This vertex, provided it exists, is that vertex $i \in V \setminus R$ for which $E(R \cup \{i\})$ maximizes the number of yet uncovered edges of E.

Two features of CGM are of special interest to us. The first one is the fact that multiple calls to CGM might generate distinct ECC solutions. Accordingly, one may call it multiple times and keep the best ECCP solution thus obtained, as we will do later on in this section. The other is that it expands its cliques with vertices that maximize the covering of yet uncovered edges. In combination with the previous feature we mentioned, this makes CGM an attractive instrument for

sampling cliques for R-SCP-ECCP formulations, as we will do in Chapter 4.

Algorithm 2 *findACliqueOf*

Require: A graph $G = (V, E)$ and an uncovered edge $\{u, v\} \in E$
Ensure: A **maximal** clique containing the vertices u and v
1: $R \leftarrow \{u, v\}$
2: $P \leftarrow N(u) \cap N(v)$
3: $z \leftarrow \text{ExtractNode}(P)$
4: **while** $z \neq$ null **do**
5: $\quad R \leftarrow R \cup \{z\}$
6: $\quad P \leftarrow P \cap N(z)$
7: $\quad z \leftarrow \text{ExtractNode}(P)$
8: **end while**
9: **return** R

3.3 The experimental framework

We will now describe the framework we used for carrying out the computational experiments reported in this Chapter and also in additional Chapters of the Part I of this thesis. Due to the absence of a consolidated repository of benchmark instances for ECCP, we opted for conducting our experiments over the randomly generated connected graphs $G = (V, E)$ introduced in LUCENA *et al.* (2010). These graphs contain a number of vertices $|V| \in \{30, 50, 70, 100, 120, 150, 200\}$, and, barring an odd case where an intended graph could not be generated for $|V| = 30$, percentage densities $d \in \{5, 10, 20, 30, 50, 70\}$. Every test graph $G = (V, E)$ is identified as vA_dB, where A and B respectively denote $|V|$ and the percentage density of G. Our option for these graphs was simply to avoid extreme situations where graph densities are either very low or very high, as implied by niche applications to ECCP.

Apart from the Gurobi procedures, all the other ones we use were implemented by us, in the C programming language. GCC version 11.2 was used to compile them, under optimization flags –O3 and –march=native. Experiments were conducted on a computer equipped with an Intel Core i7-11700K processor of 8 physical cores and 64GB RAM. The operating system used was Ubuntu 20.04 LTS.

Finally, the main metric we use to compare the performance of our algorithms is solution quality, the smaller the ECCP solutions they produce the better. Run time, as one will note, is a metric that is only used very marginally here.

3.4 Baseline results

Run time limits are imposed on the two algorithms we will describe next and are respectively defined in either CPU time, for single thread runs, and wall time, for

multi thread runs. The algorithms are:

- CGM-X: the CGM heuristic is repeatedly called, throughout a 1,800 CPU seconds span, and the best ECCP solution obtained is kept.

- Bron-Kerbosch-Gurobi (BKG): solver Gurobi takes as an input the SCP-ECCP formulation for $G = (V, E)$ (set $\mathcal{C}$ generation by BKA), and attempts to solve it to proven optimality. Every run of Gurobi is restricted to no more than 3,600 wall time seconds. The solver is allowed to run over 15 threads, under its aggressive pre-solving and aggressive cutting-plane generation options. BKA runs demanded relatively little CPU time (always much less than 3,600 CPU seconds) are not taken into account here.

3.4.1 Table and Figures

Computational results for the CGM-X and BKG are displayed in Table C-C.1. The first three columns in the table show test graph details. Under the heading "instance", instance names, vA_dB, appear in the first column and identify the input graph $G = (V, E)$, with $A = |V|$ and B standing for the percentage density of G. The number of edges of G appears in the second column, under the heading "$|E|$", while the number of distinct maximal cliques for it is shown in the third column, under the heading "$|\mathcal{C}|$". Two additional blocks of information then follow in Table C-C.1, under the headings "CGM-X" and "BKG", respectively. A description of them follows.

The CGM-X block contains three columns. Under the heading "best", the first column displays the cardinality of the best solution obtained by CGM-X, in its multiple calls to CGM. The second one brings, under the heading "# of runs", the total number of CGM calls carried out, under our imposed CPU time limit. Finally, the third column displays, under the heading "time(s)/iterations", the average number of CPU time seconds spent in each of these CGM calls.

The final four columns in Table C-C.1 are devoted to BKG results. They respectively appear under the headings (a) "$\underline{v}(\mathcal{R}_1)$", for the LPR bounds of the SCP-ECCP formulation, (b) "$\underline{v}^+(\mathcal{R}_1)$", that identifies what we shall call the "enumeration tree LPR" (ET-LPR) and corresponds to either the optimal solution value (when the Gurobi run is not aborted) or the smallest LPR bound available at any Gurobi enumeration tree node that is still open (when the run is aborted), (c) "$\overline{v}(\mathcal{R}_1)$", for the best upper bound attained by Gurobi, under our imposed run time limit, and (d) "time(s)", for BKG wall time. An entry $\sim$ for $\underline{v}(\mathcal{R}_1)$, $\underline{v}^+(\mathcal{R}_1)$, $\overline{v}(\mathcal{R}_1)$ or time(s) indicates that the corresponding value could not be computed due to hardware limitations. An entry "-" under the heading time(s) indicates that the run was aborted

due to run time limit. Finally, note that $\overline{v}(\mathcal{R}_1)$ either corresponds to κ, when BKG run time limit is not exceeded or else to an upper bound on that value, when it happens otherwise.

Formulation SCP-ECCP for any of our test graphs $G = (V, E)$ involves $|E|$ rows and $|\mathcal{C}|$ columns. Accordingly, they range from $|E| = 44$ and $|\mathcal{C}| = 40$, for instance v30_d10, to $|E| = 13,930$ and $|\mathcal{C}| = 87,326,813$ for v200_d70. Accordingly the ECCP instances we address range from very small to massive ones. As one will note, our test instances quickly become very challenging to Gurobi, with the increase of $|V|$ or the increase of graph density.

In order to make it easier for the reader to compare the results displayed in Table C-C.1, we rely on Figure 3.1, which is displayed in the main text body. The figure gives a pictorial indication of how fast the performance of CGM-X and BKG degrades with the increase of $|V|$ or the increase of graph density. The outer semicircle in it corresponds to "best" CGM-X values. Inner semicircles contain radius that are multiples of 10% smaller than the outer semicircle radius. Additionally, a distinct ray is associated in the figure to every different instance in our test set and values $\overline{v}(\mathcal{R}_1)$ and $\underline{v}(\mathcal{R}_1)$ are displayed for them over their corresponding rays. It should be noted that $\kappa \geq \lceil \underline{v}(\mathcal{R}_1) \rceil$ necessarily applies. Furthermore κ must be less or equal to the minimum between $\overline{v}(\mathcal{R}_1)$ and the best solution attained by CGM-X. Boldface instance names in Figure 3.1 indicate that BKG obtained certified optimal solutions for the corresponding instance.

3.4.2 Evaluation of CGM-X and BKG results

The multiple CGM calls implemented by CGM-X returned optimal or near optimal solutions for instances with density below 20% or fewer than 70 vertices. However, as Figure 3.1 indicates, poor quality solutions are returned by the procedure for instances that do not comply with these requirements.

Figure 3.1 shows that BKG failed to find certified optimal solutions for 17 out of our 41 test instances. Furthermore, values $\lceil \underline{v}(\mathcal{R}_1) \rceil$ and $\overline{v}(\mathcal{R}_1)$ for 14 of these 17 instances came out too far apart from each other and therefore little can be said about their optimal solution values. These 17 instances are our most challenging ones and this fact is also highlighted by the relatively poor performance CGM-X attains for them. A conclusion that stems from the fact that the BKG upper bounds for these instances, $\overline{v}(\mathcal{R}_1)$, clearly dominate their CGM-X counterparts. Finally note that Gurobi could not find any feasible solution to instances v120_d70, v150_d70 and v200_d70. The SCP-ECCP formulations for them are so large that not enough RAM was available to upload them to Gurobi.

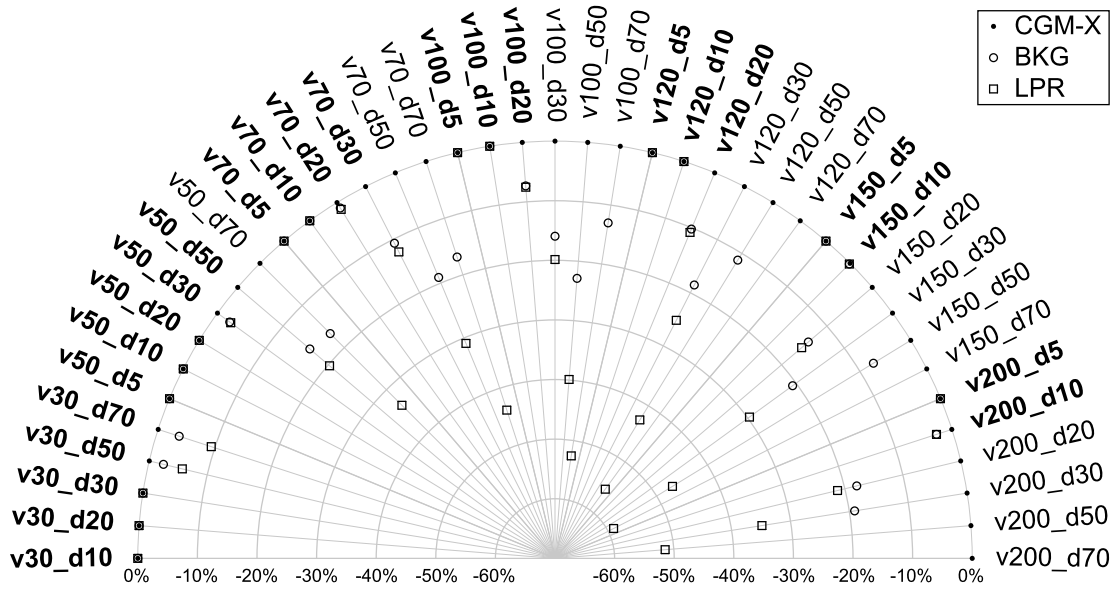

Figure 3.1: ECCP baseline results.

Chapter 4

Clique Sampling Heuristic (CSH)

The computational experiments presented in Chapter 3 reveal a clear performance dichotomy in SCP-ECCP formulations. These formulations demonstrate computational tractability for smaller instances and those with sparse edge densities, while presenting significant algorithmic challenges as graph size and density increase. In terms of number of vertices, graphs with up to 50 vertices appear to define the limit for what one may call an *empirically easy to solve instance* in our test bed, even so instance v50_d70 could not be solved to proven optimality by Gurobi in our imposed run time limit. As for graph density, the denomination would apply to instances with densities of 30% or less. Additionally, one may also suggest that Gurobi's feasible solution values for the largest and densest instances that remain unsolved, i.e., the upper bounds on κ they define, although better than those attained by CGM-X, are likely to be loose. A justification for that stems from the relatively small number of nodes the solver managed to enumerate for them, within our imposed run time limit for it.

The experiments in Chapter 3 also indicate that CGM-X has a somewhat similar behaviour to BKG, attaining very good quality solutions, i.e., optimal or near optimal ones, for the instances Gurobi attains certified optimal solutions and loose ones for those that Gurobi fails to do so, looser still than Gurobi's. On the bright side, however, we also noticed that CGM-X, in its multiple calls to CGM, managed to generate all (or almost all) different cliques of $\mathcal{C}$, for every single instance in our test bed. Such a performance appears to indicate that CGM qualifies as an attractive clique sampling instrument for designing R-SCP-ECCP formulations. With that objective in mind, we then decided to set CGM-X to make respectively 3, 30, 300 and 3000 distinct calls to CGM and generated R-SCP-ECCP formulations defined by all distinct maximal cliques identified in each of these CGM-X runs. These R-SCP-ECCP formulations (a) involve a relatively small number of variables, (b) are more likely to be solved to proven optimality by Gurobi and (c) will hopefully return better quality ECCP solutions than the best overall ECCP solution directly

obtained by CGM-X or by Gurobi, when the solver fails to return proven optimal solutions. This section is essentially devoted to exploring and testing this idea.

We call the *Clique Sampling Heuristic* (CSH) our combination of CGM-X, used as a clique sampling instrument, and Gurobi, for solving the R-SCP-ECCP formulations generated via CGM-X. As before, every run of Gurobi is restricted to no more than 3,600 wall time seconds and the solver is allowed to run over 15 threads, under its aggressive pre-solving and aggressive cutting-plane generation options.

4.1 R-SCP-ECCP Formulations

Our R-SCP-ECCP formulations originate from subsets of maximal cliques $\mathcal{C}' \subseteq \mathcal{C}$, sampled for $G = (V, E)$ by CGM-X, as previously defined. Given that CGM returns feasible solutions to ECCP, $\mathcal{C}'$ is thus guaranteed to contain feasible solutions to the problem.

Let $J' \subseteq J$ and $J'_e \subseteq J_e$ be sets of indices that are in a one-to-one correspondence with the cliques of $\mathcal{C}'$ and those cliques $C \in \mathcal{C}'$ for which $G[C]$ contains edge $e \in E$, respectively. Ideally, as previously indicated, $n' = |\mathcal{C}'|$ should be made conveniently small.

To distinguish SCP-ECCP formulation (3.3) from the R-SCP-ECCP formulations we will now describe, clique variables $\boldsymbol{y} \in \mathbb{R}^{n'}$ are associated with the latter ones. We also associate to them a polyhedral region $\mathcal{R}_2$, described as follows:

$$\sum_{j \in J'_e} y_j \geq 1, e \in E \tag{4.1}$$

$$y_j \geq 0, j \in J'. \tag{4.2}$$

A R-SCP-ECCP formulation to ECCP is then defined as

$$\text{minimize} \left\{ \sum_{j \in J'} y_j : \mathbf{y} \in \mathcal{R}_2 \cap \mathbb{Z}^{n'} \right\}, \tag{4.3}$$

with a corresponding LPR given by

$$\text{minimize} \left\{ \sum_{j \in J'} y_j : \mathbf{y} \in \mathcal{R}_2 \right\}. \tag{4.4}$$

4.2 Computational Results for CSH

Computational results for CSH are displayed in Table C-C.2. The table essentially follows the same nomenclature that applies to Table C-C.1. However, given that

four R-SCP-ECCP formulations are available for every test instance, statistics and results for each of them are presented in sequence, from the smallest to the largest formulation, separated by "/". In more detail, the number of cliques for each formulation appears under the heading $|\mathcal{C}'|$ and the upper bounds Gurobi attains for them appear under the heading $\overline{v}(\mathcal{R}_2)$. Bearing in mind the fact that Gurobi fails to find certified optimal solutions for various of ours R-SCP-ECCP formulations, the entries under the heading $\overline{v}(\mathcal{R}_2)$ will be defined so as to highlight this fact. Accordingly, two entries "b-c" are then used for these instance-formulation pairs, b standing for the formulation's rounded up ET-LPR bound and c for the best upper Gurobi attained for it. Note, in this case, that an optimal solution for the pair is an integer in the range $[b, c]$. Otherwise, when Gurobi attains a certified optimal solution to the formulation, a single entry "a", would identify its value.

As in Section 3.3, we also present here a pictorial illustration of the results obtained. They appear in Figure 4.1, follow the same nomenclature that applies to Figure 3.1 and, apart from bounds $\underline{v}(\mathcal{R}_1)$, contain all other results displayed in Figure 3.1. A difference here is that the outer semicircle in Figure 4.1 does not correspond to "best" CGM-X values, as it applies to Figure 3.1. Indeed it has a radius 10% larger than it and is necessary because a few of our instance-formulation pairs either have optimal solutions values (or Gurobi upper bounds on these values) that are above their corresponding CGM-X values. This applies, in particular, to R-SCP-ECCP formulations that originate from CGM-X runs involving 3 CGM calls, only. Finally, we will denote our different versions of CSH as CSH-3, CSH-30, CSH-300 and CSH-3000, where CSH imposes CGM-X runs involving respectively 3, 30, 300 and 3000 calls to CGM.

As one may appreciate from the results displayed in Figure 4.1, CSH appears to be a good compromise solution between the almost invariable massive SCP-ECCP formulations of Chapter 3, that could not be properly addressed by Gurobi, and empirically more tractable R-SCP-ECCP formulations. In particular, they appear to indicate that CGM is a very attractive sampling instrument for setting-up R-SCP-ECCP formulations.

For only 11 out of our 41 test instances, optimal solution values for 3-CGM-calls R-SCP-ECCP formulations are above their CGM-X counterparts in Section 3.3. The corresponding figures for the {30,300}-CGM-calls R-SCP-ECCP formulations is 2. Additionally, for only one of our 3000-CGM-calls R-SCP-ECCP formulations, optimal solution values are above their corresponding CGM-X values in Section 3.3. Furthermore, optimal values for R-SCP-ECCP formulations originating from 30 plus CGM calls, frequently proved competitive with BKG upper bounds. Finally, with the increase of graph density, R-SCP-ECCP formulations become massive, already for 30 or 300 CSH calls. Accordingly, Gurobi returns poor quality upper bounds for

them.

As a concluding note for this section, note that, although it is more likely to happen, there is not guarantee that a R-SCP-ECCP formulation associated with CSH-x, for x=A, has a better optimal solution value than its counterpart for x=B, where A>B. Take the v70_d50 instance as an example. CSH-300 returns an ECCP solution of value 126 for it while CSH-3000 returns one of value 127. However, for this particular instance, Gurobi fails to find certified optimal solutions for any of our 4 CSH generated R-SCP-ECCP formulations. Therefore, we can not conclude that the CSH-300 R-SCP-ECCP formulation has a better optimal solution value than its CSH-3000 counterpart. In addition, this example also highlights the fact that, for the instances in our test bed, R-SCP-ECCP formulations originating from CSH-x fast become far too hard for Gurobi. Accordingly, any fine tuned CSH heuristic is bound to require a good quality SCP heuristic for solving R-SCP-ECCP formulations, instead of relying on a MIP solver for that purpose.

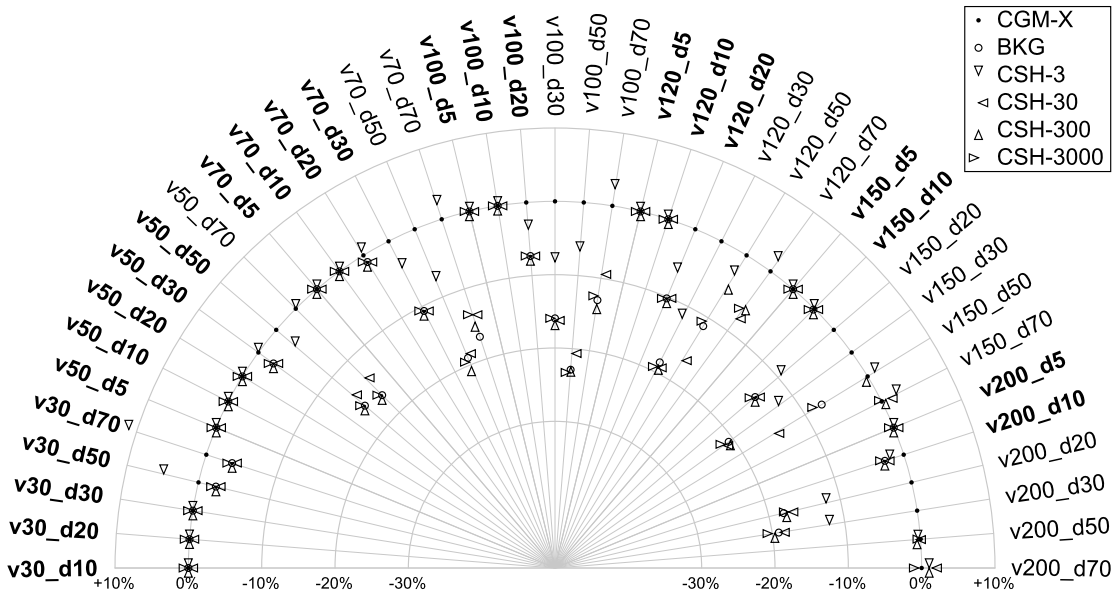

Figure 4.1: CSH results.

Chapter 5

Reduced-Cost Sampling Heuristic (RCSH)

The computational analysis in the previous chapter demonstrates the fundamental trade-off between formulation completeness and computational tractability in SCP-ECCP approaches. This chapter advances beyond the idea of using clique sampling to generate R-SCP-ECCP formulations. In particular, we will describe clique sampling conducted through the standard *Column Generation Approach* (CGA), as implemented in BP algorithms (DESROSIERS e LÜBBECKE, 2011), to achieve more principled selection of maximal cliques.

In accordance with CGA, an initial, small, R-SCP-ECCP formulation is established out of SCP-ECCP formulation to ECCP and additional variables are added to it, only as they become necessary, via an auxiliary problem. The term "necessary" applies here to any of the exponentially many variables in the SCP-ECCP formulation that (a) are currently absent from the R-SCP-ECCP formulation and (b) have the potential for improving its LPR solution, if appended to it (i.e., have a negative reduced cost for that LPR solution). Therefore, clique sampling under CGA is reduced cost oriented and is ideally carried out up to the point where no attractive variable remains outside the R-SCP-ECCP formulation. Upon reaching this point, one then attempts to solve the resulting R-SCP-ECCP formulation to proven optimal integrality and thus obtain a feasible ECCP solution, hopefully of good quality. We call this overall procedure the Reduced Cost Sampling Heuristic (RCSH).

As suggested in the previous paragraph, RCSH involves three basic steps that must be carried out in sequence, with the sequence defined by the first two steps possibly iterating. In the first step an optimal solution is obtained for the LPR of the R-SCP-ECCP formulation in hand. In the second, the optimal dual values thus obtained are used to identify, provided they exist, attractive columns(variables) that are currently absent from the R-SCP-ECCP formulation. An attractive vari-

able, in this case, is one which belongs to the SCP-ECCP formulation and has a negative reduced cost, under the dual values available. These variables are typically identified by solving an auxiliary problem, called the Pricing Problem (PP), which corresponds, in the ECCP case, to the Maximum Edge Weight Clique Problem (MEWCP) PARK *et al.* (1996). In doing so, the variable with the most negative reduced cost, provided one exists, is then selected to enter the R-SCP-ECCP formulation. This two-step ordered sequence should iterate up to the point where either no more attractive columns exist or else a run time limit is exceeded. Finally, RCSH moves on to its third and final step, where the R-SCP-ECCP formulation one is left with is attempted to be solved to proven optimality.

For the computational results reported in this chapter, PP is solved by simple inspection. This results from our use of BKA to enumerate, beforehand, all maximal cliques in $\mathcal{C}$. Proceeding in this way fits in well with the objective of the Part I of this thesis, which is essentially to identify research opportunities in the design of exact and heuristic procedures for solving ECCP, under the SCP-ECCP formulation to the problem. Fine tuning these procedures into competitive algorithms will be left as a suggestion for future work.

The chapter follows with a formal description of PP and the computational experiments carried out for RCSH.

5.1 The Pricing Problem

Ignoring for the moment the fact that SCP-ECCP formulation (3.3) is only solved exactly for the smallest or sparser graphs $G = (V, E)$ in our test bed, assume that $\{\alpha_e \geq 0 : e \in E\}$ are the dual variables associated with the inequalities in (3.1). The dual problem corresponding to (3.4) is then formulated as

$$\text{maximize} \left\{ \sum_{e \in E} \alpha_e : \boldsymbol{\alpha} \in \mathcal{R}_3 \right\}, \tag{5.1}$$

where polyhedral region $\mathcal{R}_3$ is defined as

$$\sum_{e \in E(C_j)} \alpha_e \leq 1, \ j \in J \tag{5.2}$$

$$\alpha_e \geq 0, \ e \in E. \tag{5.3}$$

Now assume that an optimal solution $\overline{\mathbf{x}}$ is available for (3.4), i.e., to the LPR of (3.3), with optimal values $\overline{\boldsymbol{\alpha}}$ applying to its corresponding dual variables. A reduced

cost r_j, for variable $x_j, j \in J$, is then computed as

$$r_j = 1 - \sum_{e \in E(C_j)} \overline{\alpha}_e \tag{5.4}$$

and, given that $\overline{\mathbf{x}}$ and $\overline{\boldsymbol{\alpha}}$ are assumed to be optimal, $r_j \geq 0$ must then hold for (5.4).

Faced with the exponentially many cliques of $\mathcal{C}$, attempts to obtain an optimal solution, $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{\alpha}})$, to (3.4), would normally involve the use of CGA (see LÜBBECKE e DESROSIERS (2005), for details). To implement it, assume that a suitable R-SCP-ECCP formulation is available for G. Namely, a formulation which originates from a *conveniently small* subset of maximal cliques $\mathcal{C}' \subseteq \mathcal{C}$ and contains at least one feasible ECCP solution for G. In addition, assume that its LPR, i.e., (4.4), is solved and $\overline{\boldsymbol{y}}$ is an optimal solution for it. Additionally, assume that $\overline{\boldsymbol{\beta}}$ are the optimal dual values for the inequalities in (4.1).

Moving forwards with CGA, a properly defined PP (see LÜBBECKE e DESROSIERS (2005), for PP details) should now be formulated and solved. Such a problem asks for the identification of a clique $C^* \in \mathcal{C}$ with $\sum_{e \in E[C^*]} \overline{\beta}_e$ as large as possible (respectively, LPR reduced cost $(1 - \sum_{e \in E[C^*]} \overline{\beta}_e)$ as small as possible). Accordingly, it corresponds to a Maximum Edge Weight Clique Problem (MEWCP) defined over $G = (V, E)$, under edge weights$\{\overline{\beta}_e : e \in E\}$. A problem that is, in general, hard to solve (see HOSSEINIAN *et al.* (2017), for complexity results associated with MEWCP).

In order to formulate PP, associate variables $\{h_i \in [0, 1] : i \in V\}$ with the vertices of $G = (V, E)$ and variables $\{z_e \geq 0 : e \in E\}$ with its edges. Additionally, consider a polyhedral region $\mathcal{R}_4$, defined as

$$h_i + h_j \leq 1,\ e = \{i, j\} \notin E \tag{5.5}$$

$$z_e \leq h_k,\ e = \{i, j\} \in E,\ k \in \{i, j\} \tag{5.6}$$

$$z_e \geq h_i + h_j - 1,\ e = \{i, j\} \in E \tag{5.7}$$

$$0 \leq h_i \leq 1,\ i \in V \tag{5.8}$$

$$z_e \geq 0,\ e \in E. \tag{5.9}$$

A formulation for PP, i.e., a formulation for MEWCP, is then given by

$$\text{maximize} \left\{ \sum_{e \in E} \overline{\beta}_e z_e : (\boldsymbol{h}, \boldsymbol{z}) \in \mathcal{R}_4 \cap (\mathbb{Z}^{|V|}, \mathbb{R}^{|E|}) \right\}. \tag{5.10}$$

After solving PP, assume that $(1 - \sum_{e \in E[C^*]} \overline{\beta_e})$ is negative, i.e., that under dual values $\overline{\boldsymbol{\beta}}$, C^* corresponds to a clique with the most negative reduced cost. Note in

this case that C^* must necessarily belong to $\mathcal{C} \setminus \mathcal{C}'$. Accordingly, one should then update $\mathcal{C}' := \mathcal{C}' \cup \{C^*\}$ and enlarge the R-SCP-ECCP formulation with a variable corresponding to C^*. Next, in a new round of our previous operations, the LPR for our updated R-SCP-ECCP formulation, (4.4), should be solved and variables with negative reduced costs should be sought for it, via the solution of a new PP.

Conversely, if $(1 - \sum_{e \in E[C^*]} \beta_e)$ results non negative, one must then conclude that there exists no clique in $\mathcal{C}$ whose corresponding variable might further reduce the LPR value in (4.4). Accordingly CGA would then come to a halt and $\overline{\boldsymbol{y}}$, after a proper mapping to the variables in $\boldsymbol{x}$, is guaranteed to correspond to an optimal LPR solution to SCP-ECCP formulation (3.4). Additionally, if $\overline{\boldsymbol{y}}$ is also integer, it would imply an optimal (integral) solution to ECCP. Finally, if $\overline{\boldsymbol{y}}$ is optimal for (4.4) but is not integer, one should then devise a branching rule that, in combination with PP, results in BP algorithm for solving ECCP exactly.

As we generate beforehand all maximal cliques $\mathcal{C}$ for every graph $G = (V, E)$ in our test bed, we opted for solving any PP we come across in our experiments, via inspection. Accordingly, no PP is explicitly formulated and solved as a MEWCP. In practice, this is not the path one would normally follow in a BP algorithm for solving ECCP but it suffices for the purposes of this thesis.

5.2 Computational experiments for RCSH

As indicated previously, RCSH implements CGA and then attempts to solve to proven integer optimality the R-SCP-ECCP formulation that thus results. Columns for this formulation are generated by firstly solving its corresponding LPR, with Gurobi's Simplex Algorithm. Instead of carrying out CGA up to the point where no attractive column remains outside the R-SCP-ECCP formulation, we impose a wall time limit of $2,000$ seconds for this clique sampling phase of RCSH. Upon exiting from a CGA run, the R-SCP-ECCP formulation one is left with is then attempted to be solved to proven optimal integrality with MIP solver Gurobi. Every run of the MIP solver is restricted to no more than 3,000 wall time seconds and it is allowed to run over 15 threads, under its aggressive pre-solving and aggressive cutting-plane generation options.

Computational results for RCSH are displayed in Table C-C.3 and the nomenclature it uses is partially borrowed from Tables C-C.1 and C-C.2. Therefore, we will only present here the additional nomenclature that remains to be described.

Every RCSH run starts with a call to CGM and a R-SCP-ECCP formulation that originates from the columns (variables) in the ECCP solution thus obtained. The number of variables in this formulation is displayed in Table C-C.3, under the heading "$|\mathcal{C}'|$". Additional variables are then generated for it, via CGA, and the

number variables in our final R-SCP-ECCP formulation appears under the heading "$|\mathcal{C}'_+|$". The ET-LPR bounds attained by the R-SCP-ECCP formulations with $|\mathcal{C}'_+|$ variables appear under the heading $\underline{v}(\mathcal{R}_2^+)$. Complementing this information, if this number is accompanied by the symbol "*", it indicates that the CPU time limit was hit by CGA and therefore attractive variables (columns) still remain outside the formulation. As an example, the $\underline{v}(\mathcal{R}_2^+)$ entry for instance v100_d70 is 83.6^* and variables with reduced costs at least as small as -0.386, which was its last most negative reduced cost obtained, are left out of the formulation. We also display the optimal value for our initial R-SCP-ECCP formulation, generated out of the CGM solution, under the heading "$\overline{v}$(CGM)" (note, in this case, that $\overline{v}$(CGM) entries invariably correspond to the number of cliques in the CGM solution).

The CPU time CGA spends on any of our test instances is displayed under the heading "CGA(t)". In complement to it, the wall time Gurobi spends (while attempting to solve the very last R-SCP-ECCP formulation generated by CGA) is displayed under the heading "R-SCP-ECCP(t)". In either case, whenever the respective run time limit is hit prior to fully accomplishing the intended goal, an entry "-" is used to indicate this outcome. For the instance v100_d50, for example, note that "-" appears under the heading "R-SCP-ECCP(t)" but not under the "CGA(t)" one. Therefore no attractive column remains outside the very last R-SCP-ECCP formulation generated by CGA. However, Gurobi fails to find a proven optimal solution for it, under our imposed wall time limit. Given the duality gap that remains to be closed by Gurobi, at the moment the run was aborted, an optimal solution value for this particular R-SCP-ECCP formulation falls within the $[163, 215]$ range and is quite likely to be much smaller than 215.

As for our previous tables, results in Table C-C.3 are also pictorially displayed in an auxiliary figure, Figure 5.1, in this case. As one may conclude from the information displayed in Figure 5.1, RCSH performs better than CGM-X but worse than BKG and the best CSH output. In particular, out of our 41 test instances, BKG generates better ECCP solutions than RCSH for 11 of them, with the reverse occurring for 6 instances (mostly, extremely large instances for which BKG could barely cope with). The two algorithms generate the same upper bounds for the remaining 24 instances. Corresponding figures for the best CSH outputs and RCSH are respectively 17, 1 and 23.

The relatively weak performance of RCSH against BKG and CSH stems primarily from its focus on improving LPR bounds for R-SCP-ECCP formulations via the reduced cost criterion. Its R-SCP-ECCP formulations can be divided into two types: (a) those for which CGA was successfully completed, i.e., no column with a negative reduced cost was found, and (b) those for which CGA was interrupted by time limit. For most of the type (a) formulations where the optimal solution is close to the

lower bound, i.e., the gap between the LPR solution and the upper bound solution is reasonably low, the columns RCSH generates are likely to be part of an optimal solution. In these cases, the solutions that RCSH reaches are either optimal or near-optimal. On the other hand, for the type (a) formulations with larger gaps, it is possible that optimal solution values might not be close to LPR bounds and, when this is indeed the case, the columns RCSH generates are unlikely to be optimal ones. Despite producing significantly fewer columns than BKG and even less than CSH-{$\geq$ 30}, RCSH seems to generate low quality columns, i.e., column that are not part of any ECCP optimal solutions. Accordingly, the optimal solution values for the R-SCP-ECCP formulations RCSH would then assemble are bound to be loose. For the case of type (b) formulations, RCSH is severely affected by hardware limitations, when dealing with some of the instances with density higher than 30% and 100 plus vertices, thus preventing it from successfully completing its CGA runs. Consequently, their R-SCP-ECCP formulations end up with fewer columns than desired (despite being already large, in terms of computational tractability) and optimal or near optimal ECCP solutions are bound to be absent from them. This combination of factors creates a scenario where RCSH performs poorly, sometimes even yielding worse results than CGM-X.

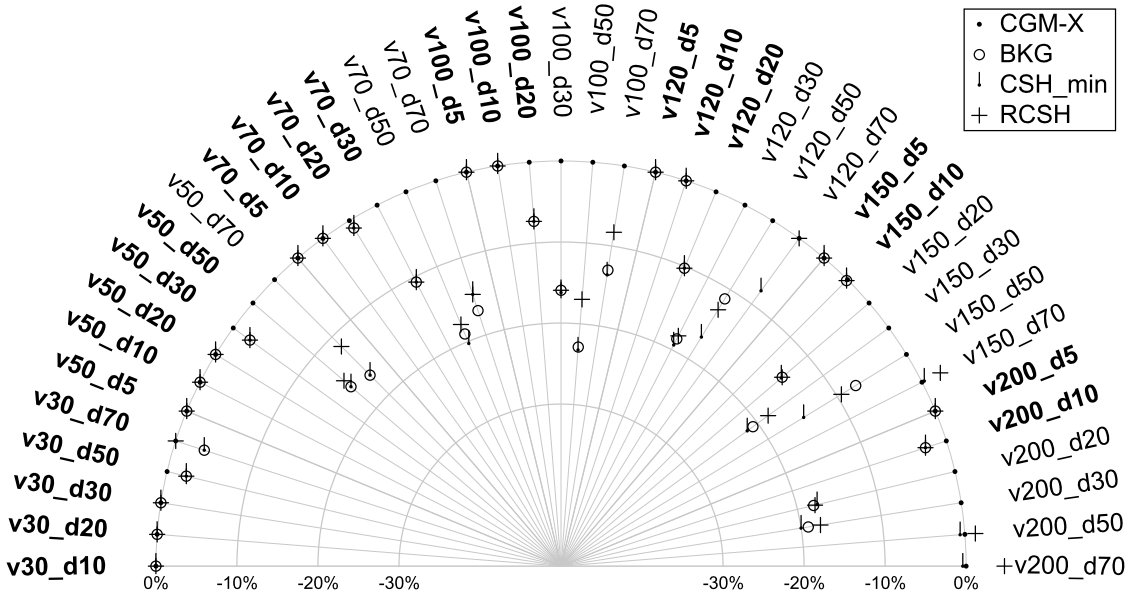

Figure 5.1: RCSH results.

Chapter 6

Enhancing RCSH with Local Search

Having established the RCSH in the previous chapter, we now extend this approach through the integration of a Local Search (LS) procedure. The resulting framework, termed LS-RCSH, is implemented, as one shall see, considering LS as a column generation procedure. In doing so, LS attempts to generate additional attractive columns to R-SCP-ECCP formulations, past the point RCSH, as implemented in Chapter 5, allows. By enlarging these formulations with columns that are LS wise attractive, we will not improve their LPR values but, due to the larger number of variables they end up containing, will stand a better chance of obtaining improved optimal integral solutions for them.

As indicated in Chapter 5, CGA generates a R-SCP-ECCP formulation defined over variables $\{y_j : j \in J'\}$, with no (reduced cost wise) attractive variable remaining outside it, i.e., belonging to $\{y_j : j \in J \setminus J'\}$. An optimal or near optimal solution to this formulation is then computed and defines the output for RCSH. We will call this solution the RCSH *Baseline Solution* (BLS). For our LS procedure we will further restrict this R-SCP-ECCP formulation with a BLS implied *soft fixing inequality* (SFI), thus obtaining what we will call a SF-SCP-ECCP formulation. More specifically, SFI enforces that a given *percentage* of the BLS variables must be part of any feasible solution to the SF-SCP-ECCP formulation and, consequently, to its LPR as well. Furthermore, optimal solutions to this LPR formulation allow us to resume CGA, targeting, this time, at the neighborhood SFI defines around BLS. Variables identified in this way are then appended to J', giving rise to an enlarged R-SCP-ECCP formulation. Finally, optimal or near optimal solutions for this enlarged R-SCP-ECCP formulation might result better than BLS.

6.1 A detailed description for LS-RCSH

Assume that the final R-SCP-ECCP formulation assembled by RCSH corresponds to (4.3) and, consequently, that its LPR is given by (4.4). Assume as well that an

optimal or near optimal solution is attained for (4.3) and $\{y_j : j \in J^*\}$, $J^* \subseteq J'$, are the nonzero variables in it. Out of J^* one then defines a SFI

$$\sum_{j \in J^*} y_j \geq \lceil \zeta * |J^*| \rceil, \tag{6.1}$$

$\zeta \in [0, 1]$, which is appended to the R-SCP-ECCP formulation to obtain a SF-SCP-ECCP formulation. It should be noted that any feasible solution to this SF-SCP-ECCP formulation contains at least $\lceil \zeta * |J^*| \rceil$ variables indexed by J^*. As an example, setting ζ to 0.9 ensures that least 90% of the variables in $\{y_j : j \in J^*\}$ are contained in any feasible solution. A description of the SF-SCP-ECCP formulation is presented next.

Let $\mathcal{R}_4$ be a polyhedral region defined as follows:

$$\sum_{j \in J'_e} y_j \geq 1, e \in E \tag{6.2}$$

$$\sum_{j \in J^*} y_j \geq \lceil \zeta * |J^*| \rceil \tag{6.3}$$

$$y_j \geq 0, j \in J'. \tag{6.4}$$

A SF-SCP-ECCP formulation is then given by

$$\text{minimize} \left\{ \sum_{j \in J'} y_j : \mathbf{y} \in \mathcal{R}_4 \cap \mathbb{Z}^{|J'|} \right\} \tag{6.5}$$

and its LPR is defined as

$$\text{minimize} \left\{ \sum_{j \in J'} y_j : \mathbf{y} \in \mathcal{R}_4 \right\}. \tag{6.6}$$

Let us now go back to (4.3), the R-SCP-ECCP formulation that gave rise to (6.5). Given that (4.3) is assumed to be the final R-SCP-ECCP formulation assembled by RCSH, no variable in $\{y_j, j \in J \backslash J'\}$, is, by definition, attractive for its corresponding LPR. However, they might be attractive to (6.6), the LPR of (6.5). We will then attempt to identify attractive columns to (6.6), via CGA, as previously carried out for RCSH. In order to do so, assume that an optimal solution $\boldsymbol{y} = \overline{\boldsymbol{y}}$ is available for (6.6). Additionally, assume that $(\boldsymbol{\beta}, \gamma)^T = (\overline{\boldsymbol{\beta}}, \overline{\gamma})^T$ is a dual optimal solution for it, where $\boldsymbol{\beta} \geq \mathbf{0}$ and $\gamma \geq 0$ are the dual variables respectively associated with inequalities (6.2) and (6.3). Reduced costs for variables y_j, $j \in J^*$, are then defined

as

$$r_j = 1 - \gamma - \sum_{e \in E(C_j)} \overline{\beta}_e \tag{6.7}$$

and must be all non negative, as ensured to any variable in $\{y_j : j \in J'\}$ by the optimality of $(\overline{\boldsymbol{y}}, \overline{\boldsymbol{\beta}}, \overline{\gamma})$. Conversely, reduced costs for variables y_j, $j \in J \setminus J'$, which are not directly affected by γ, are defined as

$$r_j = 1 - \sum_{e \in E(C_j)} \overline{\beta}_e \tag{6.8}$$

and might eventually be negative. Therefore, if one keeps $\overline{\gamma}$ aside and formulates MEWCP just exactly as we have done in (5.10), i.e., relying only on dual values $\overline{\boldsymbol{\beta}}$, it would suffice to identify negative reduced cost variables in $\{y_j : j \in J \setminus J'\}$, provided they exist. If this is the case, the procedure should iterate, carrying out a new CGA round. In order to do so, J' is firstly enlarged with the indices for the negative reduced cost variables one has identified. Accordingly, SF-SCP-ECCP formulation (6.5) is thus updated and optimal dual values for its LPR, (6.6), must then be computed. Out of these optimal duals, attractive columns for SF-SCP-ECCP are then attempted to be identified once again. CGA rounds like the one we have just described should be carried out, recursively, for as long as they are successful in enlarging J'. In the two paragraphs that follow we will describe how LS-RCSH deals with the two possible outcomes for these recursive CGA runs: namely, (a) J' is enlarged and (b) J' remains, otherwise, unaltered.

If J' is enlarged, R-SCP-ECCP formulation (4.3) should be updated with it. Next, it should be solved to optimal or near optimal integrality. If the ECCP solution thus obtained improves upon BLS, BLS should be updated with it and LS-RCSH is resumed with a new SFI and its corresponding SF-SCP-ECCP formulation. Alternatively, assume that J' increases but BLS remains unaltered. One should then reduce the right-hand-side (rhs) term in (6.1). In doing so the search neighborhood imposed by this updated SFI is made larger than before. For instance, one may initialize a CGA round with $\zeta = 0.9$ and reduce its value in 0.05 steps, whenever required. If $\zeta = 0$ results from these ζ value reductions, LS-RCSH is then exited, since SFI would thus become redundant for SF-SCP-ECCP formulation (6.5). Otherwise, CGA is resumed, under the conditions imposed by the updated value ζ. In doing so, one would be essentially moving back to the scenario described in the previous paragraph.

Now assume that J' remains unaltered. As such, CGA is necessarily carried out for just a single round and fails to find negative reduced cost variables in $\{y_j : j \in J \setminus J'\}$. One should then simply reduce the rhs term in (6.1), as described in the previous paragraph. In doing so, LS-RCSH should be resumed, implementing a CGA

round from the updated SF-SCP-ECCP formulation that thus results. Accordingly, this would take us back to the scenario described in the paragraph immediately before the previous one.

As one should note LS-RCSH is required to solve, to optimal or near optimal integrality, R-SCP-ECCP formulations with an increasing number of variables. However, for the larger or denser graphs in our test bed, these R-SCP-ECCP formulations fast become exceedingly large to be solved exactly by Gurobi. For the computational experiments in the following subsection we then decided to settle for a compromise solution. Namely, an alternative in which R-SCP-ECCP formulations are replaced with restricted forms of them. Solving these restricted R-SCP-ECCP formulations will still provide us with valid ECCP solutions and they should be, in practice, less demanding for Gurobi to solve. In summary, we will be betting on attaining better ECCP feasible solutions by solving looser formulations, hopefully to proven optimality, as opposed to poorly solving tighter ones.

To describe the idea, assume we have just finished a sequence of CGA rounds where $|J'|$ increased. Additionally, let $\overline{\zeta}$ be the ζ value for our final SF-SCP-ECCP formulation. The R-SCP-ECCP formulation one must then solve is the one that directly applies to J' or, in an alternative description, is equivalent to the SF-SCP-ECCP formulation with ζ set to 0, thus making its SFI redundant. Accordingly, setting ζ to a value in $(0, \overline{\zeta})$ gives rise to a constrained version of R-SCP-ECCP formulation we will attempt to solve. For our computational experiments, we settled for using $\zeta = \overline{\zeta} - 0.1$. In doing so we gain access to a restricted version of R-SCP-ECCP that might contain ECCP feasible solutions for which $\lceil(\overline{\zeta} - 0.1) * |J^*|\rceil \leq \sum_{j \in J^*} y_j < \lceil |J^*| \rceil$ applies and that, hopefully, improve upon the ECCP solution defined by J^*.

The LS-RCSH procedure just described in this section is summarized in Algorithm 3, providing the reader with a clear and concise overview of its key steps. Historically, a similar algorithm to LS-RCSH was first proposed in the year 2008 by Abilio Lucena and Carlos Henrique Sabóia, in a research project for CEPEL, the Research Center of Eletrobras, the Brazilian government company in charge of electricity generation and transmission. It was suggested in connection with a formulation with exponentially many variables for the long term planning of the expansion of power generation and transmission. This procedure became publicly available in the doctoral thesis of Carlos Henrique Sabóia SABÓIA (2013), supervised by Abilio Lucena. More recently, since the year 2021, Abilio Lucena has been investigating the Domatic Number Problem (DNP) CHANG (1994) and devised a BP algorithm, a LS-RCSH heuristic and a formulation by representatives CAMPÊLO *et al.* (2005) for the problem.

Algorithm 3 *LS-RCSH Algorithm for SF-SCP-ECCP*

Require: Graph $G = (V, E)$, initial parameters.
Ensure: Final solution to the R-SCP-ECCP formulation.

```
Initialize J' and BLS using CGM(G) solution
Execute RCSH for J' to update both J' and BLS
ζ ← 0.95
while ζ > 0 do
    repeat
        Solve SF-SCP-ECCP LPR for J' with SFI(BLS, ζ)
        Obtain the optimal dual solution (β̄, γ̄)
        Formulate MEWCP using dual values β̄ and solve it
        Add the variables j ∈ J \ J' with negative reduced costs to J'
    until No negative reduced cost variables were found
    if New variables were added to J' then
        Solve SF-SCP-ECCP to optimal integrality for J' with SFI(BLS, ζ − 0.1)
        if Solution improves BLS then
            Update BLS
        else
            ζ ← ζ − 0.05
        end if
    else
        ζ ← ζ − 0.05
    end if
end while
return BLS
```

6.2 Computational results for LS-RCSH

In our computational experiments for LS-RCSH, every run of the procedure was restricted to 20,000 wall time seconds, divided as follows: (a) every CGA round is limited to up to 2,000 wall time seconds and, (b) whenever J' increases, Gurobi is allowed up to 1,200 wall time seconds for attempting to solve our proposed restricted formulation for the resulting R-SCP-ECCP formulation, i.e., its corresponding SF-SCP-ECCP formulation with ζ value set to $\overline{\zeta} - 0.1$. The solver is allowed to run over 15 threads with its "aggressive presolve" and "aggressive cutting plane generation" options activated. PPs within CGA rounds are solved by inspection, with at most one attractive variable in $\{y_j : j \in J \setminus J'\}$ allowed into J', for every PP solved. Finally, every CGA round is initialized with a SFI for which $\zeta = 0.95$ applies and, whenever necessary, ζ values are reduced by 0.05 at a time.

The computational results attained by LS-RCSH are presented in Table C-C.4. Part of the nomenclature which applies to this table was already discussed for previous tables and, therefore, we will limit ourselves to explaining the missing parts.

For every test graph $G = (V, E)$, columns labeled $|\mathcal{C}1|$, $|\mathcal{C}2|$, and $|\mathcal{C}3|$ respectively indicate the number of maximal cliques (subset of variables of $\{y_j : j \in J\}$) for three

relevant R-SCP-ECCP formulations associated with LS-RCSH: (a) an initial one defined by a CGM solution, (b) the one for which the LPR of SCP-ECCP is hopefully attained (i.e, when no more attractive variables remain for it in $\{y_j : j \in J \setminus J'\}$), and (c) the very last one assembled by LS-RCSH.

For case (a) in the previous paragraph, it should be noted that $|\mathcal{C}1|$ also corresponds to the number of cliques (variables) in the ECCP solution returned by CGM. On the other hand, upper bounds on the optimal R-SCP-ECCP formulation values for cases (b) and (c) are respectively displayed in columns labeled "$\bar{v}_1(\mathcal{R}_2)$" and "$\bar{v}_2(\mathcal{R}_2)$". Additionally, the column labeled "$\underline{v}(\mathcal{R}_2)$" displays LPR values for the final R-SCP-ECCP formulations attained by RCSH and is borrowed from Table C.3. Furthermore, when entries for column labeled $\underline{v}(\mathcal{R}_2)$ are followed by *, this indicates that CGA rounds were interrupted by time limit and attractive variables y_j, $j \in J \setminus J'$, might still exist for the instance. In turn, entries under the heading "i_1" indicate the number of times BLS is improved by LS-RCSH. Finally, complementing this information, entries under the label "i_2" indicate the total number of LS-RCSH enforced ζ reductions.

Likewise previous tables, a pictorial representation of the results in Table C-C.4 are also displayed in an accompanying figure, Figure 6.1, in this case. For comparison purposes, this figure also displays CGM-X, BKG, and CSH results. In particular these CSH results correspond to the best ones attained by either CSH-3, CSH-30, CSH-300 or CSH-3000 and appear in the figure under the label "CSH_min".

Prior to discussing the quality of the results obtained by LS-RCSH, it is worth mentioning that, as it also applies to BKG and RCSH results, they are affected by the fact that some of the R-SCP-ECCP formulations (or relaxed versions of them) LS-RCSH must solve, fast become out of reach for Gurobi. For the 10 graphs in our test bed with $|V| \leq 50$ and graph densities equal or bellow 50%, where this is less of a problem, LS-RCSH lagged behind BKG and CHS_min for 2 instances, drawing with them for the other 8 instances. However, for these 2 particular instances, i.e., v30_d70 and v50_d50, the LS-RCSH solution is only 1 unit above the optimal.

For the remaining 14 *easier to solve* instances, i.e., those with certified optimal solutions available for them, they all have $|V| \geq 50$ and graph densities not above 30%. LS-RCSH matched the performances of BKG and CSH_min for them and generated optimal solutions in all cases.

The 17 *harder to solve* instances in our test bed, i.e., those for which certified optimal solutions are still unavailable, all have $|V| \geq 50$ and densities that are not bellow 20%. For 4 of them the LS-RCSH performance lagged behind the one which applies to one of their competitors. For 12 of the remaining 13 instances, the reverse occurred. Finally, for the final remaining instance, LS-RCSH matched the best performance attained by its competitors.

Of particular interest to us is the LS-RCSH performance for the v100_d20 instance. One should note that RCSH, which may be understood as a preliminary phase of LS-RCSH, generates 625 columns for it. The optimal LPR value for the R-SCP-ECCP formulation implied by these columns is 328.2 while the optimal integral value for the formulation is 330. Therefore the optimal ECCP solution for this instance must be either 329 or 330. One must stress that no better solution than 330 is capable of being raised from the current R-SCP-ECCP formulation. Proceeding from this R-SCP-ECCP formulation, LS-RCSH updates it with 29 SF-SCP-ECCP-wise attractive columns and brings the optimal integral solution for the resulting updated R-SCP-ECCP formulation to value 329. Accordingly, the ECCP upper bound returned by LS-RCSH is thus certified as optimal for the instance. This example is quite illustrative of the benefits LS-RCSH brings about.

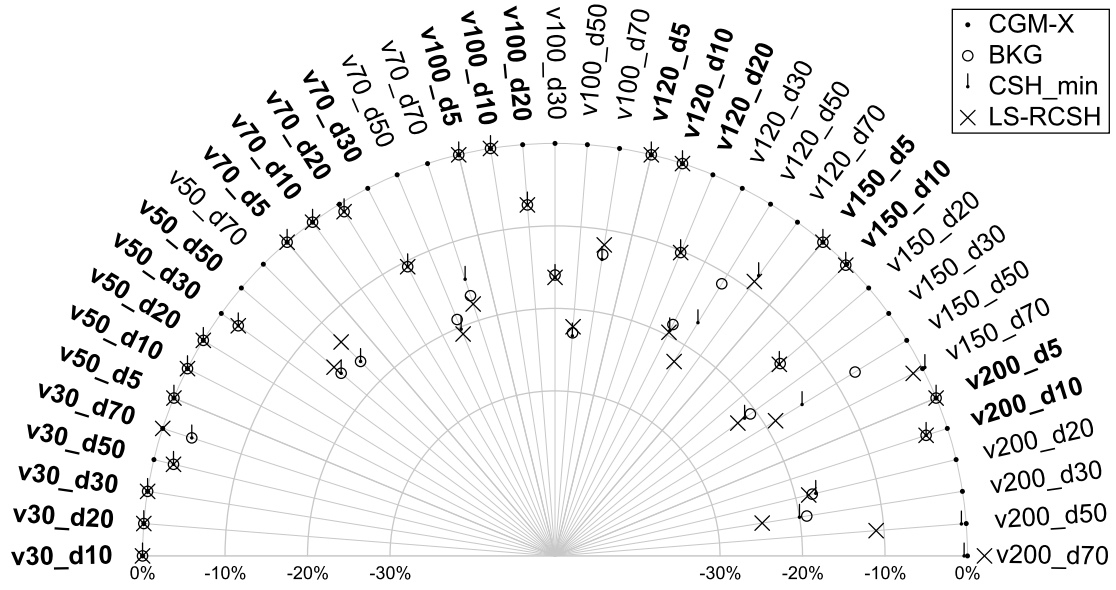

Figure 6.1: LS-RCSH results.

Chapter 7

Conclusions and Transition to Part II

The Part I of this thesis is focused on frameworks for designing algorithms, exact or heuristic, for solving SCP-ECCP, the SCP formulation to ECCP. Our aim was not so much on offering fine tuned algorithms for solving this formulation. Rather than doing that, we investigated some of the algorithmic opportunities and challenges that come with that formulation.

The computational experiments we conducted reveal that SCP-ECCP formulations are empirically easy to solve for the smaller or sparser instances in our test bed. The reverse occurs for those instances with a larger number of vertices or higher densities. As graph complexity increases, two main obstacles emerge that appear to limit the effectiveness of direct SCP approaches for ECCP.

So far, no exact solution BP algorithm exists for ECCP and devising one is certainly a very interesting research proposition. From the results obtained in this investigation, it appears quite evident that at least two main obstacles must be overcome to devise an empirically effective algorithm of this type. One is that the heuristics currently available for ECCP, used as a stand alone module, do not look very attractive. This remark is supported by the results we obtained with the CGM-X procedure, tested in Part I. For some of the harder to solve instances in our test bed, the best ECCP solution returned by CGM-X (even after performing thousands of calls to the CGM heuristic of CONTE *et al.* (2016), the current option of choice for an ECCP heuristic at the time of this writing) was quite poor. Indeed CGM-X lagged well behind all other ECCP heuristic frameworks we investigated here, including CSH-30x, which uses CGM-X as a clique sampling device to generate small attractive R-SCP-ECCP formulations.

The other obstacle we envisage stems from the empirical evidence we got, indicating that duality gaps between LPR bounds for SCP-ECCP formulations and optimal ECCP solutions appear to increase very fast with the increase of graph size or density. In this case, the formulation would require the use of cutting planes, to reinforce it and thus reduce the implicit enumeration effort. However, no obvious

cutting plane candidates appear to exist.

Despite these challenges, our investigation reveals some promising directions for exact algorithm development. Out of the results we obtained for LS-RCSH, this heuristic appears particularly suitable to become part of an ECCP BP algorithm. This conclusion stems from the fact that, throughout BP implicit enumeration, an ECCP BP algorithm would store all distinct maximal cliques it generates and, as a result of that, increasingly attractive R-SCP-ECCP formulations thus become available to LS-RCSH. Indeed, the preliminary results the first author already obtained with a BP algorithm that implements this idea to DNP, strongly supports its use for other problems.

Turning the focus once again to heuristics, one should note that although we tried to prove optimality to all distinct IP problems we come across in our frameworks, there is ample space to solve them via heuristics. For example, the PPs (i.e., MEWCPs) one is required to solve, within the RCSH and LS-RCSH frameworks, might be left in charge of MEWCP heuristics. Likewise the solution of R-SCP-ECCP formulations, as demanded by all frameworks we considered, might be carried out by unit-cost SCP heuristics.

It is worth noting that within the matheuristic landscape for MIP, the Relaxation Induced Neighborhood Search (RINS) by DANNA *et al.* (2005) represents an important development, which uses the intersection of incumbent and continuous relaxation solutions to define promising neighborhoods for local search within MIP solvers. While our LS-RCSH approach shares the principle of using the information from the best incumbent solution to guide neighborhood construction, it specifically addresses the challenge of maintaining column generation consistency throughout the search process. We were not aware of RINS at the time of developing LS-RCSH, and this parallel development represents an independent convergence toward effective neighborhood definition strategies in MIP. Interestingly, incorporating RINS-style neighborhoods into the LS-RCSH framework would be straightforward to test and could potentially yield interesting hybrid approaches, though this remains a topic for future investigation.

Finally, the CSH framework proved quite effective in assembling small R-SCP-ECCP formulations that contain good quality ECCP solutions. The best trade-off in that respect coming from the CSH-30x version of it. Accordingly, one might then consider the use of it as a warm starter to RCSH and LS-RCSH. Namely, to generate good quality initial R-SCP-ECCP formulations for them.

The insights gained from Part I establish both the potential and the limitations of heuristic approaches based on SCP formulations. While these methods achieve good performance on a significant portion of practical instances, the fundamental scalability challenges revealed by our analysis motivate the development of more

theoretically robust exact methods. The rapid growth in formulation size, the deteriorating quality of linear programming bounds, and the absence of obvious cutting plane strategies collectively point toward the need for fundamentally different algorithmic approaches that can maintain tractability across the full spectrum of problem difficulty.

Part II addresses these limitations by departing from direct SCP formulations in favor of a graph transformation approach that enables the application of branch-and-price methodology with theoretically sound branching strategies. By converting ECCP instances into equivalent Vertex Clique Cover Problems through systematic graph transformation, we overcome the structural obstacles that prevent effective branching in direct SCP formulations while maintaining the column generation advantages identified in Part I.

Part II

Branch-and-Price Algorithm for ECCP via Graph Transformation

Chapter 8

Introduction to Part II

Part I established the effectiveness of sampling-based approaches for generating high-quality restricted SCP-ECCP formulations while simultaneously revealing fundamental limitations that prevent the development of robust exact algorithms using direct SCP formulations. The rapid deterioration of linear programming relaxation bounds with increasing graph complexity, combined with the absence of effective cutting plane strategies, necessitates a fundamentally different algorithmic approach for exact solution methods.

Part II addresses these limitations by introducing a novel branch-and-price framework based on systematic graph transformation. Rather than working directly with edge covering formulations, we convert ECCP instances into equivalent Vertex Clique Cover Problems (VCCP) via this transformation. This resolves a critical structural obstacle that has prevented effective branch-and-price implementations for ECCP: traditional variable-based branching strategies destroy the pricing problem structure throughout the enumeration tree, creating heterogeneous subproblems that cannot be solved efficiently using a unified pricing algorithm (DESROSIERS e LÜBBECKE, 2011).

The graph transformation approach enables the adaptation of proven branching strategies from the graph coloring literature, specifically the branching framework developed by MEHROTRA e TRICK (1996). By reformulating ECCP as VCCP, we preserve the Maximum Weight Clique Problem structure in all pricing subproblems across the entire enumeration tree, ensuring algorithmic coherence and maintaining the theoretical foundations necessary for effective branch-and-price implementation.

While VCCP could theoretically be approached through graph coloring techniques applied to complement graphs, we develop the VCCP formulation directly to explore previously unexamined theoretical territory. This approach yields two distinct algorithmic contributions: a complete branch-and-price framework with novel branching strategies tailored to transformed graph properties, and an exact pricing algorithm that solves Maximum Vertex Weighted Clique Problems (MVWCP)

through representative-based enumeration techniques.

To provide comprehensive algorithmic comparison and emphasize the inherent computational complexity of exact methods, we develop a parallel contribution: a novel VCCP formulation based on representatives. This mathematical framework, absent from current literature, enables systematic benchmarking of our branch-and-price approach while emphasizing important characteristics of problem difficulty across varying structural properties of transformed graphs.

Part II is organized into three chapters that systematically develop and evaluate these exact methodologies. Chapter 9 presents the complete branch-and-price framework, detailing the graph transformation, adapted branching strategy that maintain pricing problem consistency, and an exact approach for the MVWCP based on representatives. Chapter 10 introduces the representative-based VCCP formulation with its associated reduction strategies and branch-and-cut techniques. Chapter 11 provides extensive computational experiments that compare all proposed exact methods against a baseline approach, analyzing both solution quality and computational tractability across diverse instance classes while identifying the boundaries of practical applicability for each algorithmic approach.

Chapter 9

Branch-and-Price Formulation and Methodology

This chapter presents a branch-and-price framework for the Edge Clique Cover Problem (ECCP). The main challenge in developing such a framework lies in the formulation of appropriate branching strategies that maintain the structural integrity of the pricing problem across all nodes in the enumeration tree.

Direct application of branch-and-price to ECCP encounters significant difficulties due to the complex constraints that branching decisions impose on the pricing subproblems. These constraints fundamentally alter their optimization structure, leading to heterogeneous pricing problems throughout the search tree that require distinct solution approaches.

We address this challenge through the conversion of ECCP into its equivalent Vertex Clique Cover Problem (VCCP) formulation via graph transformation. This transformation enables the development of robust branching strategies that preserve the pricing problem structure across all tree nodes.

Within the VCCP framework, we present the mathematical foundation necessary for the algorithmic implementation, including a specialized exact algorithm for the Maximum Vertex Weighted Clique Problem (MVWCP) that serves as the pricing subproblem. The resulting framework provides a theoretically sound and computationally viable approach to exact ECCP solution while establishing a foundation for further algorithmic development in this domain.

9.1 Motivation

9.1.1 The Branching Rule Problem

The central obstacle in developing a robust Branch-and-Price algorithm for ECCP lies in the formulation of appropriate branching rules. Traditional variable-based

branching strategies, while conceptually straightforward, fundamentally compromise the algorithmic framework's structural integrity.

Consider a standard branching approach applied to a fractional solution $\bar{\boldsymbol{x}}$ of the linear programming relaxation. Upon identifying a fractional variable $\bar{x}_i$, conventional branching creates two subproblems with constraints:

$$\text{Branch 1:} \quad x_i = 1 \tag{9.1}$$

$$\text{Branch 2:} \quad x_i = 0 \tag{9.2}$$

The constraint $x_i = 1$ forces the inclusion of clique C_i in the solution, which can be accommodated through straightforward formulation modifications. The master problem constraints remain structurally unchanged, and the pricing problem can incorporate this requirement through appropriate objective function adjustments.

However, the constraint $x_i = 0$ creates fundamental structural problems that cascade throughout the algorithmic framework. Excluding clique C_i requires not only removing the corresponding column from the current formulation but also modifying the pricing problem to prevent the regeneration of this forbidden clique. This modification destroys the clique structure that underlies the pricing problem's formulation.

The resulting pricing subproblems throughout the enumeration tree become fundamentally different optimization problems, each with distinct constraint structures and solution characteristics. This structural heterogeneity forces the development of more complex procedures that were difficult to establish effectively at the time of this research.

9.1.2 Robust Branching via Graph Transformation

The resolution of these branching difficulties emerges through a particular reconceptualization of the ECCP solution space. Rather than branching directly on clique variables, we develop a robust branching strategy based on the systematic transformation of ECCP into an equivalent Vertex Clique Cover Problem (VCCP) defined on a specially constructed graph $K(G) = (\bar{V}, \bar{E})$.

This transformation, originally introduced by Kou et al. KOU *et al.* (1978), establishes a bijective correspondence between edge clique covers in the original graph and vertex clique covers in the transformed graph. In this transformation, each edge $e = \{a, b\} \in E$ of the original graph G corresponds to a vertex $v_e \in \bar{V}$ in the transformed graph. Two vertices v_{e_i} and v_{e_j} in $K(G)$ are connected by an edge if and only if the corresponding edges e_i and e_j in G can be jointly covered by some clique.

More formally, let $I_{i,j} \subset V$ denote the set of endpoints of edges e_i and e_j. An edge $\{v_{e_i}, v_{e_j}\} \in \bar{E}$ exists if and only if $I_{i,j}$ forms a clique in G. This construction ensures that each edge in the original graph corresponds to exactly one vertex in $K(G)$, and adjacencies in $K(G)$ reflect the potential for joint coverage by cliques in the original graph.

The key innovation lies in the adoption of a branching from MEHROTRA e TRICK (1996) adapted to this graph-theoretic framework. Instead of branching on variable values, we branch on the structural relationships between vertex pairs in $K(G)$. Specifically, given a fractional solution with clique C_j containing vertices $v_{\{a,b\}}$ and $v_{\{c,d\}}$, we create branches based on their coverage relationship:

$$\text{Branch I (Together):} \quad \text{Vertices } v_{\{a,b\}} \text{ and } v_{\{c,d\}} \text{ must appear together in a clique} \tag{9.3}$$

$$\text{Branch II (Separated):} \quad \text{Vertices } v_{\{a,b\}} \text{ and } v_{\{c,d\}} \text{ must appear separately in cliques} \tag{9.4}$$

The *together* branch is implemented by merging vertices $v_{\{a,b\}}$ and $v_{\{c,d\}}$ into a supervertex $v_{\{a,b,c,d\}}$, with edges connecting to vertices that were adjacent to both original vertices. This operation preserves the clique structure while enforcing the constraint that these vertices must be covered by the same clique in any feasible solution.

The *separated* branch simply removes the edge between $v_{\{a,b\}}$ and $v_{\{c,d\}}$ in $K(G)$. This modification ensures that no clique can simultaneously contain both vertices, thereby enforcing their separation into different cliques.

This structural branching approach maintains the essential property of robustness: the pricing problem remains a Maximum Vertex Weighted Clique Problem defined over the modified graph $K'(G)$ at each node. The formulation structure, solution algorithms, and optimality conditions remain consistent throughout the enumeration tree. Only the underlying graph topology changes, while the fundamental optimization problem structure persists.

From the algorithmic perspective, these graph modifications naturally lend themselves to straightforward implementation strategies. The *together branch* can be implemented through vertex contraction operations that preserve adjacency relationships, while the *separated branch* requires simple edge deletion. Both operations maintain the fundamental clique structure necessary for pricing problem solution.

The mathematical framework underlying this approach establishes several critical theoretical properties. First, the bijective correspondence between edge clique covers in G and vertex clique covers in $K(G)$ ensures that optimal solutions to the

transformed problem yield optimal solutions to the original ECCP with identical objective values. Second, the branching strategy provides complete coverage of the solution space, guaranteeing that optimal solutions can be found through systematic enumeration.

Furthermore, the size and structural properties of $K(G)$ remain manageable under reasonable assumptions about the original graph. As demonstrated by HEVIA *et al.* (2023), when G has degeneracy d, $K(G)$ satisfies $|\bar{V}| = m \leq dn$ and $|\bar{E}| = O(d^2 m)$, where $m = |E|$ and $n = |V|$. This polynomial relationship in the degeneracy parameter ensures practical applicability for sparse graphs commonly encountered in real-world applications.

The robust Branch-and-Price algorithm thus achieves the objective of exact optimization by preserving pricing problem structure through graph transformations. The algorithm maintains the sophisticated solution procedures developed for Maximum Vertex Weighted Clique Problems while systematically exploring the solution space through principled branching decisions. This combination of structural preservation and systematic enumeration provides the algorithmic foundation necessary for solving ECCP instances to proven optimality.

9.2 The Graph Transformation

We present the graph construction that transforms ECCP to VCCP by converting edges into vertices while maintaining the clique structure. This transformation enables us to apply vertex-based algorithms to solve the edge clique cover problem through a bijective correspondence between solutions.

9.2.1 Definition and Construction of $K(G)$ Graphs

Given an input graph $G = (V, E)$ for the ECCP, we construct the graph $K(G) = (\bar{V}, \bar{E})$ through a systematic vertex-edge correspondence that preserves the essential clique structure. The construction proceeds as follows.

For each edge $e = \{a, b\} \in E$, we create a corresponding vertex $v_e \in \bar{V}$. This establishes a bijective mapping between the edge set of the original graph and the vertex set of the transformed graph, ensuring that every edge coverage requirement in the original problem translates to a corresponding vertex coverage requirement in the transformed problem.

The edge set $\bar{E}$ is then defined to capture the relationships between edges that can be jointly covered by cliques in the original graph. Specifically, two vertices v_{e_i} and v_{e_j} are adjacent in $K(G)$ if and only if the corresponding edges e_i and e_j can be covered by a common clique in G.

More formally, let $I_{i,j} \subset V$ denote the set of endpoints of edges e_i and e_j. An edge $\{v_{e_i}, v_{e_j}\} \in \bar{E}$ exists if and only if $I_{i,j}$ forms a clique in G. This condition ensures that the adjacency structure in $K(G)$ precisely reflects the potential for joint coverage by cliques in the original graph.

The construction algorithm proceeds efficiently by examining each edge $e = \{u, v\}$ in G and identifying its common neighbors $N(e) = N[u] \cap N[v] \setminus \{u, v\}$. For each common neighbor $c \in N(e)$, the algorithm creates edges in $K(G)$ connecting the vertex representing edge $\{u, v\}$ to vertices representing edges $\{u, c\}$ and $\{v, c\}$, corresponding to the triangular clique $\{u, v, c\}$. Additionally, for any pair of common neighbors $c, d \in N(e)$ that are adjacent in G (i.e., $\{c, d\} \in E$), the algorithm creates three edges in $K(G)$ for the 4-clique $\{u, v, c, d\}$: connecting vertices representing edges $\{u, v\}$ to $\{c, d\}$, $\{u, c\}$ to $\{v, d\}$, and $\{u, d\}$ to $\{v, c\}$. This approach achieves $\mathcal{O}(|E| \cdot \Delta^2)$ complexity, where Δ is the maximum degree in G, avoiding the quadratic enumeration of all edge pairs.

This construction preserves some critical structural properties. First, the size of $K(G)$ remains manageable under reasonable assumptions about the original graph structure. As demonstrated by Hevia et al. HEVIA *et al.* (2023), when G has degeneracy d, $K(G)$ satisfies $|\bar{V}| = m \leq dn$ and $|\bar{E}| = O(d^2 m)$, where $m = |E|$ and $n = |V|$.

Second, the construction maintains the essential relationship between cliques in the original graph and cliques in the transformed graph. Every clique in G that covers a set of edges corresponds to a clique in $K(G)$ that covers the corresponding set of vertices, with identical cardinality properties.

9.2.2 Correspondence Between Problems

The main property of the graph transformation is its preservation of the covering structure through a bijective correspondence between solutions. This correspondence operates bidirectionally: every minimum edge clique cover in G induces a minimum vertex clique cover in $K(G)$ of identical cardinality, and vice versa. The following theorem formalizes this relationship.

Theorem 1. *Let $G = (V, E)$ be a graph and $K(G) = (\bar{V}, \bar{E})$ be its transformed graph. There exists a bijective correspondence between edge clique covers of G and vertex clique covers of $K(G)$ that preserves cardinality. Specifically, the minimum edge clique cover number of G equals the minimum vertex clique cover number of $K(G)$.*

Proof. Define ϕ mapping each edge clique C_i in G to vertex set $\phi(C_i) = \{v_e : e \subseteq C_i\}$ in $K(G)$, and ψ mapping each vertex clique D_j in $K(G)$ to $\psi(D_j) = \bigcup_{v_e \in D_j} e$ in G. Since edges e_1, e_2 can be covered by a common clique in G if and only if $\{v_{e_1}, v_{e_2}\} \in \bar{E}$

by construction of $K(G)$, we have: (i) $\phi(C_i)$ forms a clique in $K(G)$ because all edges in C_i share the clique C_i, (ii) $\psi(D_j)$ forms a clique in G because adjacency in $K(G)$ implies joint coverage capability, (iii) both mappings preserve coverage completeness since every edge e covered by some C_i corresponds to vertex v_e covered by $\phi(C_i)$ and vice versa, and (iv) $\psi \circ \phi = \text{id}$ and $\phi \circ \psi = \text{id}$ establishing bijection with $|\mathcal{C}| = |\mathcal{D}|$. □

During the graph transformation process, certain vertices in $K(G)$ may become isolated, that is, they have no adjacent vertices in the resultant graph. These isolated vertices correspond to edges in the original graph G that share no endpoints with any other edges, making them trivial cliques that must necessarily appear in any feasible edge clique cover. Since these isolated vertices represent mandatory components of any optimal solution, they can be removed from the optimization problem without loss of generality. Throughout this thesis, we extract these isolated vertices during preprocessing and apply our algorithms to the resulting connected components of $K(G)$. We denote by $|\overline{V}_{\text{iso}}|$ the number of isolated vertices removed from the original transformed graph. The optimal solution value for the original ECCP instance is then $|\overline{V}_{\text{iso}}| + \kappa(K(G))$, where $\kappa(K(G))$ represents the minimum vertex clique cover number of $K(G)$ after isolated vertex removal.

9.3 The SCP-VCCP Formulation for ECCP

Building upon the graph transformation, we present the complete mathematical formulation that enables the optimization of the converted problem. The formulation leverages the vertex clique cover structure to create a set covering problem with favorable computational properties.

Given a graph $K(G) = (\bar{V}, \bar{E})$ and the set $\mathcal{C}$ containing all maximal cliques related to it, we associate a variable $x_j \in [0, 1]$ to each maximal clique $C_j \in \mathcal{C}$. The binary nature of these variables reflects the discrete decision of whether to include each clique in the covering solution.

Furthermore, we identify by $J_v \subseteq J$ those cliques (variables) of $\mathcal{C}$ that cover/-contain vertex $v \in \bar{V}$. This index set structure enables the formulation of coverage constraints while maintaining the relationship between cliques and the vertices they cover.

The polyhedral region R_5 is defined by the constraint system:

$$\sum_{j \in J_v} x_j \geq 1, \quad v \in \bar{V} \tag{9.5}$$

$$x_j \geq 0, \quad j \in J \tag{9.6}$$

The coverage constraints (9.5) ensure that every vertex in $\bar{V}$ is covered by at least one selected clique, while the non-negativity constraints (9.6) maintain the feasible region structure.

The SCP-VCCP formulation for ECCP is then given by:

$$\min \left\{ \sum_{j \in J} x_j : x \in R_5 \cap \mathbb{Z}^n \right\} \tag{9.7}$$

with its corresponding Linear Programming Relaxation (LPR) defined as:

$$\min \left\{ \sum_{j \in J} x_j : x \in R_5 \right\} \tag{9.8}$$

The validity of formulation (9.7) follows directly from the graph transformation and the correspondence between problems. For any input graph $K(G) = (\bar{V}, \bar{E})$ and vertex $v \in \bar{V}$, the definition of $\mathcal{C}$ ensures that there exists at least one maximal clique $C_j \subseteq \mathcal{C}$ such that $K(G)[C_j]$ contains v.

The coverage constraints (9.5) require that each vertex in $\bar{V}$ be covered by at least one of these cliques, corresponding precisely to the requirement that each edge in the original graph G be covered by at least one clique. The objective function in (9.7) minimizes the number of cliques needed to achieve complete coverage.

Since a solution to the VCCP covers all vertices in $\bar{V}$ by definition, the bijective correspondence ensures that the corresponding cliques also cover all edges in G. Therefore, we can construct an optimal solution for ECCP with the same cardinality as that obtained for VCCP, establishing the equivalence between the problems.

9.4 The R-SCP-VCCP Formulations

While the full SCP-VCCP formulation provides theoretical completeness, practical computation requires more manageable problem sizes. The restricted SCP-VCCP formulations (R-SCP-VCCP) address this computational challenge by working with carefully selected subsets of the complete clique collection.

The restricted formulations use subsets of maximal cliques $\mathcal{C}' \subseteq \mathcal{C}$, obtained by converting ECCP heuristic solutions from G into corresponding cliques in $K(G)$. Following Conte et al. CONTE *et al.* (2016), these heuristic solutions provide initial feasible clique collections for the VCCP formulation.

Let $J' \subseteq J$ and $J'_v \subseteq J_v$ be the index sets that correspond to the cliques in $\mathcal{C}'$ and those cliques $C \in \mathcal{C}'$ for which $K(G)[C]$ contains vertex $v \in \bar{V}$, respectively. The parameter $n' = |\mathcal{C}'|$ represents the size of the restricted formulation and should be kept conveniently small to ensure computational tractability.

To distinguish the SCP-VCCP formulation (9.7) from the R-SCP-VCCP formulations, we introduce clique variables $y \in \mathbb{R}^{n'}$ associated with the restricted formulation. We also define a corresponding polyhedral region R_6, described as follows:

$$\sum_{j \in J'_v} y_j \geq 1, \quad v \in \bar{V} \tag{9.9}$$

$$y_j \geq 0, \quad j \in J' \tag{9.10}$$

The coverage constraints (9.9) maintain the requirement that every vertex in $\bar{V}$ be covered by at least one clique from the restricted collection, while constraints (9.10) define the feasible region for the restricted variables.

An R-SCP-VCCP formulation for ECCP is then defined as:

$$\min \left\{ \sum_{j \in J'} y_j : y \in R_6 \cap \mathbb{Z}^{n'} \right\} \tag{9.11}$$

with the corresponding Linear Programming Relaxation (LPR) given by:

$$\min \left\{ \sum_{j \in J'} y_j : y \in R_6 \right\} \tag{9.12}$$

The restricted formulations serve two main purposes within the branch-and-price framework. First, they provide computationally manageable problem sizes that can be solved by modern integer programming solvers. Second, they enable the implementation of column generation procedures that systematically expand the formulation with improving cliques.

The restriction process maintains feasibility guarantees through the use of heuristic solutions that ensure $\mathcal{C}'$ contains at least one feasible solution to the original problem. This property is crucial for the correctness of the column generation procedure, as it ensures that the restricted master problem remains feasible throughout the optimization process.

The relationship between the restricted and complete formulations provides theoretical bounds on solution quality. While the optimal value of the restricted formulation may exceed the optimal value of the complete formulation, the column generation procedure systematically identifies improving cliques that reduce this gap until optimality is achieved.

9.5 Restricted Master Problem

The restricted master problem (RMP) is implemented using the R-SCP-VCCP formulation (9.11) with binary variables $y_j \in \{0, 1\}$ for $j \in J'$. The RMP begins with the initial clique collection $\mathcal{C}'$ obtained from heuristic solutions and serves as the core component that evolves throughout the branch-and-price algorithm.

At each iteration of the column generation process, the algorithm solves the Linear Programming Relaxation (9.12) of the RMP to obtain an optimal primal solution $\tilde{y}$ and the corresponding optimal dual solution. The dual solution provides economic information about the marginal value of covering each vertex in $K(G)$, which becomes essential for identifying improving columns.

The RMP maintains feasibility throughout the algorithm since the initial collection $\mathcal{C}'$ contains at least one feasible solution to the original problem. As the algorithm progresses, new cliques discovered through the pricing mechanism are added to $\mathcal{C}'$, correspondingly updating the formulation.

The optimal solution of the RMP relaxation serves two critical purposes: it provides bounds on the optimal solution value, and its dual information guides the search for beneficial columns in the pricing problem, as detailed in the following section.

9.6 The Pricing Problem

The pricing mechanism enables dynamic column generation within our algorithm by identifying cliques that can improve the current linear programming relaxation. This component transforms the dual information from the restricted master problem into a systematic search for beneficial columns.

Let $\{\alpha_v \geq 0 : v \in \bar{V}\}$ be the dual variables associated with the coverage inequalities (9.9). The dual problem corresponding to the linear programming relaxation (9.12) is formulated as:

$$\max \left\{ \sum_{v \in \bar{V}} \alpha_v : \alpha \in R_7 \right\} \tag{9.13}$$

where the polyhedral region R_7 is defined as:

$$\sum_{v \in V(C_j)} \alpha_v \leq 1, \quad j \in J' \tag{9.14}$$

$$\alpha_v \geq 0, \quad v \in \bar{V} \tag{9.15}$$

The dual constraints (9.14) ensure that the total dual value associated with any clique does not exceed the corresponding primal variable coefficient, while constraints (9.15) maintain non-negativity of the dual variables.

Assume that an optimal solution $\tilde{\boldsymbol{y}}$ is available for the RMP relaxation (9.12), with optimal dual values $\tilde{\boldsymbol{\beta}}$ for the coverage constraints. The reduced cost r_j for any clique variable y_j, $j \in J'$, is calculated as:

$$r_j = 1 - \sum_{v \in V(C_j)} \tilde{\beta}_v \tag{9.16}$$

Given the optimality of $\tilde{\boldsymbol{y}}$ and $\tilde{\boldsymbol{\beta}}$, we have $r_j \geq 0$ for all $j \in J'$ by the complementary slackness conditions of linear programming.

To implement column generation, we seek to identify cliques $C^* \in \mathcal{C} \setminus \mathcal{C}'$ with the most negative reduced cost. This corresponds to finding a clique C^* that maximizes $\sum_{v \in V(C^*)} \tilde{\beta}_v$, which is equivalent to solving a Maximum Vertex Weighted Clique Problem (MVWCP) defined over $K(G) = (\bar{V}, \bar{E})$ with vertex weights $\{\tilde{\beta}_v : v \in \bar{V}\}$.

To formulate the pricing problem, we associate variables $\{h_i \in [0,1] : i \in \bar{V}\}$ with the vertices of $K(G) = (\bar{V}, \bar{E})$ and variables $\{z_e \geq 0 : e \in \bar{E}\}$ with its edges. The polyhedral region R_8 is defined as:

$$h_i + h_j \leq 1, \quad e = \{i,j\} \notin \bar{E} \tag{9.17}$$

$$z_e \leq h_k, \quad e = \{i,j\} \in \bar{E}, k \in \{i,j\} \tag{9.18}$$

$$z_e \geq h_i + h_j - 1, \quad e = \{i,j\} \in \bar{E} \tag{9.19}$$

$$0 \leq h_i \leq 1, \quad i \in \bar{V} \tag{9.20}$$

$$z_e \geq 0, \quad e \in \bar{E} \tag{9.21}$$

Constraints (9.18) ensure that an edge variable can be positive only if both endpoint vertices are selected, while constraints (9.19) force edge variables to be positive when both endpoints are selected. Together, these constraints enforce the clique structure in the solution.

The pricing problem formulation is then given by:

$$\max \left\{ \sum_{v \in \bar{V}} \beta_v h_v : (h,z) \in R_8 \cap (\mathbb{Z}^{|\bar{V}|}, \mathbb{R}^{|\bar{E}|}) \right\} \tag{9.22}$$

After solving the pricing problem, we examine the reduced cost $(1 - \sum_{v \in V[C^*]} \beta_v)$. If this value is negative, then under the current dual values β, clique C^* has negative reduced cost and can potentially improve the objective value of the RMP relaxation.

In this case, C^* must necessarily belong to $\mathcal{C} \setminus \mathcal{C}'$, and we update $\mathcal{C}' := \mathcal{C}' \cup \{C^*\}$

while expanding the R-SCP-VCCP formulation with a variable corresponding to C^*. The process then iterates with a new solution of the updated RMP relaxation and a new pricing problem.

Conversely, if $(1 - \sum_{v \in V[C^*]} \beta_v) \geq 0$, then no clique in $\mathcal{C}$ has a corresponding variable that can improve the current RMP relaxation value. The column generation phase terminates, and $\tilde{\boldsymbol{y}}$, after appropriate mapping to the original variables, corresponds to an optimal solution of the complete SCP-VCCP linear programming relaxation.

If $\tilde{\boldsymbol{y}}$ is also integer, this implies an optimal integral solution for ECCP. However, if $\tilde{\boldsymbol{y}}$ is optimal but fractional, the algorithm must employ a branching rule to continue the search for an optimal integral solution, leading to the branch-and-price framework.

9.7 Branching Strategy

The correctness of our branch-and-price algorithm depends fundamentally on a branching strategy that guarantees complete exploration of the solution space when the linear relaxation yields fractional solutions. Our approach adapts the branching rule introduced by MEHROTRA e TRICK (1996) for the graph coloring problem to the set covering formulation of the Vertex Clique Cover Problem (VCCP), while incorporating principles from GRAMM *et al.* (2006) which exploits graph structure in combinatorial branching.

The branching strategy maintains the essential property of *robustness*: the pricing problem remains a Maximum Vertex Weighted Clique Problem defined over modified versions of the graph $K(G)$ at each node, denoted as $K'(G) = (\bar{V}', \bar{E}')$. The formulation structure, solution algorithms, and optimality conditions remain consistent throughout the enumeration tree, with only the underlying graph topology changing.

9.7.1 Branching Template

Given an optimal linear relaxation solution $\tilde{\boldsymbol{y}}$ to the R-VCCP-SCP, our branching strategy follows a systematic three-step template:

1. **Vertex Selection**: Select two vertices $v_i, v_j \in \bar{V}'$ from the restricted graph $K'(G)$.

2. **Together Branch**: Enforce that vertices v_i and v_j must be covered by the same clique.

3. **Separated Branch**: Enforce that vertices v_i and v_j must be covered by different cliques.

The critical component lies in the systematic selection of the vertex pair (v_i, v_j). We adapt the structural approach from GRAMM *et al.* (2006), who select edges covered by the fewest maximal cliques for branching decisions in their combinatorial algorithm. This strategy targets the most *constrained* elements to yield more efficient enumeration trees, as edges with minimal clique coverage represent bottlenecks in the solution space.

We incorporate this constraint-based selection approach into our column generation setting. While enumerating all maximal cliques covering each vertex would be computationally prohibitive due to exponential complexity, we can efficiently approximate this property by focusing on columns currently in the R-VCCP-SCP formulation. This adaptation preserves the efficiency benefits of constraint-based selection while targeting precisely those cliques identified as relevant for improving the lower bound through the pricing mechanism.

Primary Vertex Selection

Let $\mathcal{F}(\bar{V}') \subseteq \bar{V}'$ denote the set of vertices associated with fractional variables in the optimal solution $\tilde{\boldsymbol{y}}$:

$$\mathcal{F}(\bar{V}') = \left\{ v \in \bar{V}' \mid \exists j \in J_v \text{ such that } \tilde{y}_j \in (0,1) \right\} \tag{9.23}$$

where J_v denotes the index set of columns covering vertex v.

We select the primary vertex $v_i \in \mathcal{F}(\bar{V}')$ with minimum coverage by columns in the current R-VCCP-SCP formulation:

$$I = \arg \min_{v \in \mathcal{F}(\bar{V}')} |J_v| \tag{9.24}$$

For ties ($|I| > 1$), we select $v_i \in I$ with the most fractional sum of solution variables covering it, providing the vertex closest to having equal fractional coverage. This tie-breaking criterion was chosen without extensive theoretical analysis, as the focus of our investigation lies in the overall algorithmic framework rather than fine-tuning of tie-breaking procedures.

Lemma 1. *If vertex $v_i \in \bar{V}'$ is covered by a fractional column in the optimal linear relaxation solution $\tilde{\boldsymbol{y}}$, then v_i must be covered by at least two columns in the current solution.*

Proof. Suppose vertex v_i is covered by only one column C_j with $\tilde{y}_j$ fractional. Since the covering constraints require $\sum_{j \in J_{v_i}} \tilde{y}_j \geq 1$, we have $\tilde{y}_j \geq 1$ for this single column.

Combined with $\tilde{y}_j \leq 1$, this implies $\tilde{y}_j = 1$, contradicting the fractional assumption. □

Lemma 1 ensures that if $I = \emptyset$, then $\tilde{\boldsymbol{y}}$ is integral, guaranteeing the availability of valid branching vertices for fractional solutions.

Secondary Vertex Selection

Having selected primary vertex v_i, let $\mathcal{C}_{v_i}$ denote the set of all formulation columns covering v_i. The secondary vertex v_j is chosen from:

$$v_j \in \left(\bigcup_{C \in \mathcal{C}_{v_i}} C \right) \setminus \left(\bigcap_{C \in \mathcal{C}_{v_i}} C \right) \tag{9.25}$$

This selection criterion ensures v_j has inconsistent coverage relationships with v_i in the current solution. Some covering cliques include both vertices while others cover only v_i. This inconsistency makes the pair (v_i, v_j) natural for binary branching as indicated in MEHROTRA e TRICK (1996): enforce joint coverage or prevent it. Among vertices satisfying this condition, we select v_j with minimum column coverage $|J_{v_j}|$, again applying the constraint-based selection principle from GRAMM *et al.* (2006) to target the most structurally constrained vertex. Lemma 1 guarantees the existence of such vertices by ensuring at least two distinct columns cover any fractionally covered vertex.

9.7.2 Branch Implementation

Together Branch

The Together Branch enforces joint coverage by merging vertices v_i and v_j into supervertex v_{ij}. The modified graph $K'(G) = (\bar{V}', \bar{E}')$ is constructed as:

$$\bar{V}' = (\bar{V} \setminus \{v_i, v_j\}) \cup \{v_{ij}\} \tag{9.26}$$

$$\bar{E}' = \{\{u, w\} \in \bar{E} : u, w \notin \{v_i, v_j\}\} \tag{9.27}$$

$$\cup \{\{v_{ij}, v_k\} : \{v_i, v_k\} \in \bar{E} \text{ and } \{v_j, v_k\} \in \bar{E}\} \tag{9.28}$$

The supervertex v_{ij} connects to vertex v_k if and only if both v_i and v_j were adjacent to v_k, ensuring that any clique containing v_{ij} corresponds to a clique containing both original vertices.

We update the R-VCCP-SCP formulation by eliminating columns containing

exactly one of $\{v_i, v_j\}$:

$$\mathcal{E}_{\text{together}} = \{C_k \in \mathcal{C}' : |C_k \cap \{v_i, v_j\}| = 1\} \tag{9.29}$$

Remaining columns containing both vertices are updated to contain v_{ij} instead.

Separated Branch

The Separated Branch prevents joint coverage by removing the edge between vertices. The modified graph is:

$$\bar{V}' = \bar{V} \tag{9.30}$$

$$\bar{E}' = \bar{E} \setminus \{\{v_i, v_j\}\} \tag{9.31}$$

We update the R-VCCP-SCP formulation by eliminating columns containing both vertices:

$$\mathcal{E}_{\text{separated}} = \{C_k \in \mathcal{C}' : \{v_i, v_j\} \subseteq C_k\} \tag{9.32}$$

9.7.3 Theoretical Properties

Theorem 2. *The proposed branching strategy provides complete coverage of the solution space for the VCCP. Every feasible integral solution to the original problem appears in exactly one leaf node of the enumeration tree.*

Proof. Consider any feasible integral solution $\boldsymbol{y}^*$ and vertex pair (v_i, v_j) selected at any branching node. Exactly one condition holds:

Case 1: There exists clique C_k with $y_k^* = 1$ such that $\{v_i, v_j\} \subseteq C_k$. The solution satisfies the Together Branch constraint.

Case 2: No clique C_k with $y_k^* = 1$ contains both v_i and v_j. The solution satisfies the Separated Branch constraint.

Since these cases are mutually exclusive and exhaustive, every feasible solution appears in exactly one branch. Systematic application establishes complete solution space coverage. □

The branching strategy preserves pricing problem structure throughout the enumeration tree. Both implementations yield modified Maximum Vertex Weighted Clique Problems: the Together Branch operates on the contracted version of the parent node's restricted graph $K'(G)$ with supervertex v_{ij}, while the Separated Branch operates on the parent node's restricted graph with edge $\{v_i, v_j\}$ removed. This consistency enables uniform application of Maximum Vertex Weighted Clique algorithms across all nodes.

9.8 Computational Implementation Considerations

The transition from theoretical framework to practical implementation introduces several critical considerations that significantly impact algorithm performance and correctness. These implementation aspects address the management of restricted graphs, column handling strategies, constraint validity, and formulation feasibility throughout the branch-and-bound enumeration process.

9.8.1 Restricted Graph Operations

At any node in the search tree, the pricing algorithm operates on a *restricted graph* $K'(G)$ rather than the original transformed graph $K(G)$. This restricted graph represents the original graph structure modified by the branching constraints accumulated along the path from the root to the current node.

The restricted graph $K'(G) = (\bar{V}', \bar{E}')$ is constructed by systematically applying the constraint pool $\mathcal{B}$ to the original transformed graph. For *separated* constraints, the corresponding edges are removed from $\bar{E}$ to prevent forbidden vertex pairs from appearing in the same clique. For *together* constraints, vertices are merged into supervertices using union-find data structures, with adjacency relationships updated accordingly.

This graph modification process ensures that any clique found in $K'(G)$ automatically satisfies all branching constraints imposed at the current node. The pricing problem structure remains that of a Maximum Vertex Weighted Clique Problem, but the underlying graph topology reflects the accumulated branching decisions.

The dynamic nature of these graph modifications requires efficient data structures and algorithms to maintain computational tractability. Our implementation employs bit representation for the adjacency matrix, which enables very fast graph-related operations including adjacency queries, neighborhood intersections, and clique procedures through efficient bitwise operations.

9.8.2 Column Generation and Maximality Requirements

A fundamental implementation challenge arises from the requirement that columns in the R-SCP-VCCP formulation correspond to maximal cliques. This requirement creates subtle but important distinctions between cliques that are maximal with respect to the original graph versus those that are maximal with respect to the current restricted graph.

When a clique is generated through the pricing procedure at a particular node, it must be maximized within the scope of the restricted graph $K'(G)$ rather than the original graph $K(G)$. This ensures that the clique represents a maximal feasible

structure under the current branching constraints, even if it might not be maximal in the original graph.

The implication of this requirement is that columns generated at a particular node are not necessarily valid for other nodes in the search tree. A clique that is maximal under one set of branching constraints may become non-maximal or even infeasible under a different set of constraints. Therefore, unlike standard branch-and-price implementations where columns can be shared across the entire search tree, our approach restricts column sharing to parent-child relationships. This constraint significantly impacts the column management strategy and requires careful tracking of which columns are valid at each node, consequently slowing the convergence of lower bounds throughout the search tree since nodes cannot leverage the full column generation history accumulated across the entire enumeration process.

9.8.3 Column Management Under Branching Constraints

Column Exclusion

Each time a new node is created through branching, the associated branching constraint must be incorporated into the R-SCP-VCCP formulation. This incorporation process requires systematic examination of existing columns to identify those that violate the new constraint.

For *separated* constraints that forbid vertices $v_{\{a,b\}}$ and $v_{\{c,d\}}$ from appearing in the same clique, any existing column containing both vertices must be removed from the formulation. For *together* constraints that require these vertices to appear in the same clique, any column containing exactly one of these vertices (but not both) must be removed.

The column exclusion process can potentially remove critical columns that provide coverage for specific vertices in the graph $K(G)$. This removal may result in some coverage constraints becoming infeasible, creating a situation where certain vertices in $\bar{V}$ cannot be covered by any remaining column in the current formulation.

The systematic identification of constraint violations requires efficient data structures to track which vertices appear in which columns. The implementation must maintain these relationships dynamically as columns are added and removed throughout the search process.

Feasibility Restoration

When column exclusions result in uncovered vertices, the formulation becomes infeasible and requires immediate correction. The feasibility restoration mechanism addresses this situation by generating emergency columns that provide the necessary coverage while respecting all current branching constraints.

For each uncovered vertex $v \in \bar{V}$, the algorithm creates a singleton clique containing only that vertex. This singleton is then maximized within the restricted graph $K'(G)$ to obtain the largest possible clique that includes v while satisfying all branching constraints.

The maximization procedure operates through a greedy expansion process. Starting from the singleton $\{v\}$, the algorithm iteratively adds vertices that are adjacent to all vertices currently in the clique, continuing until no more vertices can be added while maintaining the clique property under the current graph restrictions.

This feasibility restoration process ensures that the R-SCP-VCCP formulation remains feasible at every node, enabling the continued application of column generation and branching procedures. While these emergency columns may not represent optimal choices, they provide the minimum coverage necessary for algorithmic correctness.

9.8.4 Search-tree Exploration Mechanism

Lets call an *exploration step* as the algorithmic cycle systematically executed for advancing the search-tree exploration. The adopted *exploration step* consists of the following stages:

1. selection of an idle node to be explored.
2. branching into children nodes.
3. column generation for each node until optimal bound.
4. pruning tests execution.
5. node's storage for future selection

Selection of an idle node to be explored. The node's selection is performed via breadth-first search. The node with the worst optimal bound is retrieved from a heap data structure, which has $O(1)$ time complexity for extraction task. The local optimal bound is attained when the column generation solves the pricing problem to its optimality and certify it cannot produce any column with negative reduced cost anymore.

Branching into children nodes. The branching happens following the braching rule discussed before.

Column generation for each node until optimal bound. The column generation is executed for each child node. The optimal bound is attained when no negative reduced cost column cant be produced anymore, guarrantedly.

Pruning tests execution. At this stage, the algorithm performs bound and integrality pruning tests. Pruning by the bound happens when the local optimal bound is higher than an incumbent solution. Pruning by integrality happens when the linear relaxation finishes presenting an integer solution.

Node's storage for future selection. Once both nodes are processed, they are stored in a heap structure that works as our idle node's manager. The heap is maintained with the min-heap property, which has worst time-complexity of $O(\log n)$.

9.9 An Exact Pricing Algorithm Based on Representatives

To enhance computational efficiency in the presence of branching constraints, we developed an exact pricing algorithm that operates over the restricted graph $K'(G)$ that has already been modified to respect all constraints imposed by the current branch of the search tree. This approach systematically explores the clique space through a representative-based enumeration strategy followed by a maximization and deduplication phase.

The pricing algorithm implements a two-phase approach to efficiently identify improving columns. The first phase uses vertex representatives to systematically explore restricted subgraphs and collect promising cliques. The second phase maximizes these cliques and removes duplicates to produce the final set of columns for addition to the master problem.

This design provides computational advantages by focusing the expensive clique enumeration on smaller subgraphs while ensuring that all generated columns are maximal and distinct. Algorithm 4 provides the complete procedural framework for this approach.

9.9.1 Representative-Based Enumeration Strategy

The first phase uses vertex representatives to partition the search space and collect promising cliques. For each vertex $i \in \bar{V}'$ in the restricted graph $K'(G)$, we designate i as a representative and construct a further restricted subgraph G'_i based on neighborhood, ordering, and dual value criteria.

For representative i, we include vertex j in G'_i if and only if: (1) $j \in N[i]$ (j is in the closed neighborhood of i), (2) $j \geq i$ in the vertex traversal, and (3) j has positive dual value $\beta_j > 0$. This corresponds to line 3 in Algorithm 4.

The neighborhood constraint ensures that only vertices that can potentially form cliques with i are considered. The ordering constraint $j \geq i$ prevents the same clique from being discovered multiple times through different representatives. The dual

value constraint focuses the search on vertices that can contribute positively to the objective function.

Once the restricted subgraph G'_i is constructed, the algorithm applies the Bron-Kerbosch algorithm to enumerate maximal cliques within this subgraph (line 4). For each representative, the algorithm stores the cliques with the most negative reduced cost in a collection pool for later processing (line 5).

Algorithm 4 MWCP Solver with Representatives

Require: Restricted graph $K'(G) = (\bar{V}', \bar{E}')$, dual solution β
Ensure: Set of distinct maximal cliques with negative reduced cost
1: Initialize clique pool $\mathcal{P} \leftarrow \emptyset$
2: **for** each representative $i \in \bar{V}'$ **do**
3: Create subgraph G'_i containing vertices $\{j \in N[i] : j \geq i \text{ and } \beta_j > 0\}$
4: Find maximal cliques in G'_i using Bron-Kerbosch algorithm
5: Store cliques with most negative reduced cost in pool $\mathcal{P}$
6: **end for**
7: **for all** clique $C \in \mathcal{P}$ **do**
8: Maximize C with respect to $K'(G)$
9: **end for**
10: Remove duplicate cliques from $\mathcal{P}$, keeping only distinct ones
11: **return** $\mathcal{P}$

9.9.2 Maximization and Deduplication Phase

The second phase processes all cliques collected from the representative-based enumeration. Each clique in the pool undergoes a maximization procedure to ensure it becomes a maximal clique with respect to the restricted graph $K'(G)$ (lines 7-9 in Algorithm 4).

The maximization process expands each clique by iteratively adding vertices that maintain the clique property. This ensures that all generated columns correspond to maximal cliques, as required by the master problem formulation.

After maximization, some cliques may become identical, creating redundancy in the collection. The algorithm removes these duplicates, retaining only distinct maximal cliques (line 10). This deduplication step ensures that each improving column is unique and contributes meaningfully to the master problem.

The final set of distinct maximal cliques with negative reduced cost is then returned to be added to the RMP formulation as new columns (line 11).

9.9.3 Computational Considerations

The representative-based approach provides computational benefits through the restriction of clique enumeration to smaller subgraphs. When dual values are concen-

trated on a small subset of vertices, the restricted subgraphs G'_i become significantly smaller than the original restricted graph, leading to substantial computational savings.

The two-phase design separates the enumeration process from the maximization and deduplication operations, enabling more efficient resource utilization.

This exact pricing algorithm provides an efficient approach for the branch-and-price framework by systematically managing the exponential clique space while maintaining optimality guarantees particularly essential for our column generation algorithm.

Chapter 10

A VCCP Formulation Based on Representatives

In this chapter, we propose a novel formulation for the Vertex Clique Cover Problem (VCCP) that draws inspiration from the seminal work of YÜCEOĞLU *et al.* (2017) concerning the robust graph coloring problem. This representative-based approach provides an alternative mathematical framework for VCCP and enables comprehensive benchmarking of our Branch-and-Price algorithm. Through this formulation, we aim to gather empirical evidence regarding the computational complexity and inherent hardness characteristics of VCCP.

10.1 Mathematical Formulation

Let $G = (V, E)$ be a simple, undirected, and connected graph, where $n = |V|$ denotes the number of vertices and $m = |E|$ represents the number of edges. Two vertices i and j are considered adjacent if and only if $\{i, j\} \in E$. We define the open neighborhood of vertex i as $N(i) = \{j \in V : \{i, j\} \in E\}$. The closed neighborhood is given by $N[i] = N(i) \cup \{i\}$. Additionally, we introduce the closed in-neighborhood as $N^-[i] = \{1, 2, \ldots, i\} \cap N[i]$ and the closed out-neighborhood as $N^+[i] = \{i, i + 1, \ldots, n - 1, n\} \cap N[i]$.

For completeness, we recall that an independent set is a subset of vertices in which no two vertices are adjacent, while a clique is a subset of vertices in which every pair of vertices is adjacent.

In the context of VCCP, we introduce the concept of representative vertices. Each representative vertex serves to identify and characterize a clique that includes itself along with a subset of other vertices from its neighborhood. This representative-based approach provides a natural decomposition of the vertex set into disjoint cliques.

To formalize this concept, we define representation variables that determine the clique membership structure. For each vertex $i \in V$, we introduce binary variables x_{ij} for all vertices $j \in N^+[i]$:

$$x_{ij} = \begin{cases} 1, & \text{if vertex } i \text{ represents vertex } j \\ 0, & \text{otherwise} \end{cases} \tag{10.1}$$

Under this representation scheme, if $x_{ii} = 1$, then vertex i serves as a representative vertex for some clique; otherwise, vertex i is represented by another representative vertex within the same clique.

The VCCP formulation based on representatives is formulated as the following integer programming problem:

$$\text{minimize} \left\{ \sum_{i \in V} x_{ii} : \boldsymbol{x} \in \mathcal{R}_9 \right\}, \tag{10.2}$$

where polyhedral region $\mathcal{R}_9$ is defined as

$$\sum_{i \in N^-[j]} x_{ij} = 1, \quad \forall j \in V \tag{10.3}$$

$$x_{ij} \leq x_{ii}, \quad \forall i \in V, \forall j \in N^+[i] \tag{10.4}$$

$$\sum_{j \in I} x_{ij} \leq x_{ii}, \quad \forall i \in V, \forall I \in \mathcal{I}_i \tag{10.5}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in V, \forall j \in N^+[i] \tag{10.6}$$

where $\mathcal{I}_i$ denotes the family of maximal independent sets in the subgraph induced by $N^+(i)$.

The number of variables in the representative-based formulation exhibits polynomial growth characteristics that depend critically on the graph's structural properties. Specifically, the total number of binary variables x_{ij} is bounded by $\sum_{i \in V} |N^+[i]|$, which in the worst case approaches $O(n^2)$ for dense graphs where each vertex can represent up to $n - 1$ other vertices. However, for sparse graphs with maximum degree Δ, this bound tightens to $O(n \cdot \Delta)$, making the formulation particularly attractive for applications involving low-degree networks. The minimum degree δ provides a lower bound on formulation size, as vertices with degree δ contribute at least δ variables each.

10.2 Validity of the Formulation

The proposed formulation provides a mathematically sound representation of VCCP. We establish its validity through a systematic analysis of each constraint family.

Constraint (10.3) enforces the fundamental requirement that each vertex must be assigned to exactly one representative, thereby ensuring that every vertex belongs to precisely one clique in the solution. This uniqueness property is essential for maintaining the disjoint nature of the clique cover.

Constraint (10.4) establishes the logical relationship between representation variables and representative status. Specifically, if vertex $i \in V$ is not designated as a representative (i.e., $x_{ii} = 0$), then it cannot represent any vertex in its out-neighborhood $N^+[i]$. This constraint maintains the consistency of the representative-based structure.

Constraint (10.5) constitutes the core mechanism that guarantees the clique property for each set of represented vertices. To demonstrate this, let $C_i \subseteq N^+[i]$ represent the set of vertices represented by vertex i. Suppose, for the sake of contradiction, that C_i does not form a clique. Then there exist vertices $j_1, j_2 \in C_i$ such that $\{j_1, j_2\} \notin E$. By the definition of independent sets, there must exist a maximal independent set $\tilde{I} \in \mathcal{I}_i$ such that $\{j_1, j_2\} \subseteq \tilde{I}$. Consequently, constraint (10.5) would require:

$$x_{ij_1} + x_{ij_2} + \sum_{k \in \tilde{I} \setminus \{j_1, j_2\}} x_{ik} \leq x_{ii} \tag{10.7}$$

However, since i represents both j_1 and j_2, we have $x_{ij_1} = x_{ij_2} = 1$, which contradicts the assumption that $x_{ii} = 1$ when additional vertices are included in $\tilde{I}$. This contradiction establishes that C_i must indeed form a clique. Therefore, vertex i can represent all valid cliques, whether maximal or not, that can be formed by vertices in $N^+[i]$.

The uniqueness property enforced by constraint (10.3) ensures that the resulting cliques in any feasible solution are naturally disjoint, as required by the VCCP definition.

Constraint (10.6) imposes the integrality requirement on all decision variables.

Finally, the objective function (10.2) minimizes the number of representative vertices, which by construction corresponds to minimizing the number of cliques in the cover. Since every vertex must be either a representative or represented by some representative vertex, all vertices are guaranteed to be covered by the resulting clique collection. Thus, this formulation correctly solves VCCP.

It is important to note that constraint (10.5) does not enforce maximality on the cliques in the solution. Consequently, the optimal solution may contain disjoint

cliques that are either maximal or non-maximal.

10.3 Computational Complexity and Reduction Strategies

A critical observation regarding constraint (10.5) concerns the cardinality of the family $\mathcal{I}_i$. The maximal independent sets in $\mathcal{I}_i$ can be obtained by enumerating all maximal cliques in the complement graph of the subgraph induced by vertices in $N^+(i)$. As established in the literature TOMITA *et al.* (2006), the number of such constraints can grow exponentially with respect to the problem size, potentially rendering the formulation computationally intractable for large-scale instances.

To address this computational challenge, we propose a systematic reduction strategy that maintains the formulation's validity while significantly reducing the number of constraints. The key insight is to relax the maximality requirement for the independent set family $\mathcal{I}_i$ while preserving the essential structural properties that ensure solution feasibility.

10.3.1 Weak Reduction Strategy

Our first reduction approach focuses on a specific subfamily of independent sets. We redefine $\mathcal{I}_i$ to include only independent sets of cardinality 2 within the subgraph induced by $N^+(i)$. Note that considering independent sets of size 1 would effectively reduce constraint (10.5) to constraint (10.4), providing no additional structural information.

The validity of this reduced formulation follows from the same theoretical foundation as the complete formulation. By construction, if vertices j_1 and j_2 are non-adjacent and both represented by vertex i, then there must exist an independent set $\tilde{I}$ of size 2 containing both vertices. The corresponding constraint ensures that this configuration violates the clique property, maintaining solution validity.

Since 2-independent sets represent the minimal non-trivial independent sets that impose meaningful clique structure conditions, we refer to this variant as the *weak-reduced formulation.*

10.3.2 Strong Reduction Strategy

To enhance the computational effectiveness of the weak-reduced formulation, we propose a strengthening approach that maximizes the constraining power of the selected 2-independent sets. The methodology involves extending each 2-independent set to its maximal form within the subgraph induced by $N^+(i)$, while ensuring that

only distinct constraints are included in the formulation. We may call this new polyhedral region as $\mathcal{R}_{10}$.

An alternative approach would involve eliminating 2-independent sets that become proper subsets of larger maximized independent sets. While this would yield an even more compact formulation, our primary objective is to strengthen the linear programming relaxation bound through enhanced constraint quality rather than merely reducing formulation size. Given that the number of constraints remains bounded by the cardinality of the 2-independent set collection from the weak-reduced formulation, the additional constraints in the strong-reduced formulation represent an acceptable computational trade-off.

The strong-reduced formulation serves as our primary computational framework for the experimental investigations presented in this thesis.

10.4 Separation Heuristics for Clique Consistency Cuts

To further strengthen the reduced formulations considered, we develop a systematic approach for generating violated cuts from the inequality family given by constraint (10.5). This separation procedure enhances the linear programming relaxation by identifying and adding the most promising violated inequalities during the solution process.

10.4.1 Theoretical Foundation

Constraint (10.5) establishes that for each vertex $i \in V$, we have a family of inequalities of the form:

$$\sum_{j \in I} x_{ij} \leq x_{ii}, \quad \forall I \in \mathcal{I}_i \tag{10.8}$$

where $\mathcal{I}_i$ represents the set of all maximal independent sets in the subgraph induced by vertices in $N^+(i)$.

Each such inequality can be uniquely characterized by the pair $\{i, I\}$, where i denotes the potential representative vertex and I represents a specific independent set. For each potential representative $i \in V$, there exists a subset $\overline{\mathcal{I}}_i \subset \mathcal{I}_i$ such that for all $I \in \overline{\mathcal{I}}_i$, the constraint $\{i, I\}$ is violated by the current fractional solution.

Let $\tilde{\boldsymbol{x}}$ denote an optimal solution to the linear programming relaxation. To efficiently explore the violated constraints in $\overline{\mathcal{I}}_i$, we construct a reduced search space by defining:

$$J_i = \{j \in N^+(i) \mid \tilde{x}_{ij} > 0\} \tag{10.9}$$

This set J_i contains only those vertices in the out-neighborhood of i that have positive representation values in the current fractional solution. By focusing exclusively on vertices with positive fractional values, we significantly reduce the computational burden of the separation procedure while maintaining completeness with respect to violated constraints.

Subsequently, we generate the set $\widetilde{\mathcal{I}}_i$ containing all maximal independent sets in the subgraph induced by vertices in J_i. This allows us to construct the collection of potential violated constraint candidates, denoted by $[i, \widetilde{\mathcal{I}}_i] = \{\{i, I\} \mid I \in \widetilde{\mathcal{I}}_i\}$. Empirical evidence demonstrates that this approach yields a substantially smaller search space compared to the complete set $[i, \mathcal{I}_i]$. By construction, we can establish that $\overline{\mathcal{I}}_i \subseteq \widetilde{\mathcal{I}}_i$, thereby guaranteeing that $[i, \widetilde{\mathcal{I}}_i]$ includes all violated constraints.

10.4.2 Algorithmic Implementation

Algorithm 5 presents the main framework of our separation heuristic. For each vertex $i \in V$, the algorithm constructs the corresponding set $\widetilde{\mathcal{I}}_i$ through a systematic process.

Algorithm 5 Separation Heuristic for Clique Consistency Cuts

```
Input: x, N⁺(·), cg, strategy, V
Output: constraints

constraints ← ∅

for each i ∈ V do
    I ← ∅
    J ← ∅
    J ← getActiveOutNeighbors(i, x, N⁺(·))
    if J = ∅ then
        continue
    end if
    I ← getIndependentSetsFromSubgraph(J, cg)
    applyCuttingStrategy(strategy, constraints, i, I, x, N⁺(·))
end for

return constraints
```

The function GETACTIVEOUTNEIGHBORS implements the construction of set J_i following the rule specified in the previous section. Subsequently, the function GETINDEPENDENTSETSFROMSUBGRAPH executes a Bron-Kerbosch algorithm to enumerate all cliques in the subgraph induced by J_i within the complement graph

of G (denoted as cg in Algorithm 5). This approach leverages the fundamental relationship that a clique in the complement graph corresponds to an independent set in the original graph.

Once the set $[i, \widetilde{\mathcal{I}}_i]$ has been constructed, we can apply any desired cutting strategy to select the most promising violated constraints from this collection.

10.4.3 Cutting Strategies and Violation Measurement

To quantify the degree of constraint violation, we define the violation value for a constraint characterized by representative i, independent set I, and solution vector $\boldsymbol{x}$ as:

$$v(i, I, \boldsymbol{x}) = \sum_{j \in I} x_{ij} - x_{ii} \tag{10.10}$$

This metric measures the extent to which the current fractional solution violates the clique consistency constraint. A positive violation value indicates that the constraint is indeed violated and represents a candidate for inclusion in the strengthened formulation.

Based on this violation measure, we consider three distinct cutting strategies in the present work:

1. **Most Violated Constraints Selection**: This strategy identifies and selects the constraints with the highest violation value across all potential representatives and independent sets. This approach prioritizes the most severely violated constraints, potentially providing the strongest dual bounds.

2. **Most Violated Constraints per Representative**: This strategy selects the most violated constraints for each representative vertex $i \in V$, ensuring a balanced distribution of cuts across all potential representatives. This approach may provide better overall coverage of the constraint space.

3. **Complete Violated Constraint Selection**: This comprehensive strategy includes all identified violated constraints in the strengthened formulation. While this approach maximizes the constraint coverage, it may lead to computational challenges due to the increased problem size.

10.5 A Branch-and-Cut Framework for Experimentation

To evaluate the effectiveness of the representative-based VCCP formulation and the proposed separation heuristics, we implement a branch-and-cut algorithm that sys-

tematically applies the cutting strategies developed in this chapter. The algorithmic framework operates in two phases: an initial cutting plane phase that strengthens the linear programming relaxation through iterative addition of violated clique consistency constraints from family (10.5), followed by a branch-and-bound phase that seeks integer solutions when necessary.

The cutting plane phase begins by solving the linear programming relaxation of the representative-based VCCP formulation under the strong reduction strategy. The algorithm then applies the separation heuristics described in Section 10.4 to identify violated clique consistency constraints, adding these cuts to strengthen the formulation iteratively until no further violations can be identified or computational limits are reached. Upon termination of this phase, if the resulting solution remains fractional, the algorithm converts formulation variables to binary type and proceeds with branch-and-bound enumeration.

During the enumeration phase, additional cuts can be generated at fractional nodes through callback procedures, with specific cutting strategies determining the selection and addition of violated constraints. The framework accommodates the three cutting strategies presented in this chapter—most violated constraint selection, most violated constraint per representative, and complete violated constraint selection—enabling systematic comparison of their effectiveness across different problem characteristics.

This branch-and-cut implementation serves as the foundation for the experimental evaluation presented in Chapter 11, where each cutting strategy is systematically tested across diverse instance classes to assess both the quality of linear programming bounds and overall solution performance.

Chapter 11

Computational Experiments

This chapter presents the experimental evaluations of our algorithms for the Edge Clique Cover Problem discussed in Part II. The computational study assesses algorithmic performance across solution quality and computational efficiency through a systematic comparison of different approaches.

11.1 Computational Environment

All experiments were conducted on a dual-socket AMD EPYC 9754 system with 256 physical cores and 1.2 TiB memory. To ensure fair comparison, all algorithms were executed using single-core configuration within a Docker container environment. The system runs Linux 6.8.0-71-generic on Debian GNU/Linux 11 (bullseye). External solver integration employs Gurobi Optimizer version 12.0.1 build v12.0.1rc0 (linux64). All other algorithmic implementations were developed from scratch using the C programming language. The code compilation was performed by gcc version 10.2.1 with the optimization flag –O3.

11.2 Test Instances

The experimental evaluation employs the same instance set used in Section 3.3, consisting of randomly generated connected graphs, introduced in LUCENA *et al.* (2010), with vertices $|V| \in \{30, 50, 70, 100, 120, 150, 200\}$ and percentage densities $d \in \{5, 10, 20, 30, 50, 70\}$. Each graph is identified as vA_dB where A and B denote the number of vertices and density percentage respectively. These instances provide comprehensive coverage of problem difficulty ranging from sparse small graphs to dense large graphs, enabling thorough algorithmic assessment across a diverse structural characteristics.

11.3 Experimental Design and Methodology

We designed eight experimental scenarios to systematically evaluate the different algorithmic approaches discussed in Part II, as detailed in Table 11.1. With these eight scenarios, 3 experimentation blocks are realized:

1. *Baseline method.* This experimentation block establishes the reference values on which we will base our analysis. Here we borrow the same algorithm used in Chapter 3 which the only difference is the use of only one CPU core. We refer to it here as BKG-1. In Table 11.1, this block encompass only experiment-1.

2. *VREP best strategy.* In chapter 10 one discuss several strategies that lead to variants of a branch-and-cut approach. From Table 11.1, this experimentation block encompass all the experiments-2-7. For each of these configurations, we refer to as VREP-experiment id. In this block, we test each of them against each other following a solution quality metric.

3. *Branch-and-Price.* In this experimentation block we test our proposed Branch-and-Price algorithm against the best VREP from the second block and the BKG-1.

Table 11.1: Experimental scenarios

Exp.	**Algorithm**	**Configuration**	**Time (s)**
1	BKG-1	1-core, 0 cuts, presolve on	3600
2	VREP-2	Best-cuts + 0 cuts/node	3600
3	VREP-3	Best-cuts-per-rep + 0 cuts/node	3600
4	VREP-4	All-cuts + 0 cuts/node	3600
5	VREP-5	Best-cuts + 1 cut/node	3600
6	VREP-6	Best-cuts-per-rep + 1 cut/node	3600
7	VREP-7	All-cuts + 1 cut/node	3600
8	Branch-and-Price	1 core + SCP heuristic	3600

All of the algorithms composing the experimentations of this chapter focus on solving either the SCP-VCCP or the R-SCP-VCCP formulations. As presented in Chapter 9, these ECCP reformulations depend on the transformation of the original graph G into a graph $K(G)$, which is the input graph of the current algorithms that we are discussing the test bed now.

BKG-1 setup

BKG-1 solves the SCP-VCCP formulation. For that respect, it runs BKA in order to populate the model with all the maximal cliques and follows solving the formulation

with the commercial solver Gurobi. The solver is allowed to run under agressive pre-solving option and it is time constrained by 3600 seconds. The solver is not allowed to generate any type of cutting planes. In the majority of the instances, the time spent by BKA is relatively very small compared to 3600 seconds, so we discard tracking it.

VREP setup

From experimentation 2 to 7, we solve the VCCP formulation by representatives with a branch and cut framework. As presented in Table 11.1. Experiments 2,3,4 have the first phase of their branch-and-cut, a cutting plane phase, organized to accept, respectively, the best cuts, the best cuts per representative and all the separated cuts. The second phase, if necessary, follows with a branch and bound approach for the enumeration tree. Experiments 5,6,7 have the first phase of their branch-and-cut, respectively, identical to the one in experiments 2,3,4. They differ only in the second phase, where they employ a branch-and-cut approach with acceptation of up to 1 cut per node. Due to limitations imposed by the usage of the commercial solver, as the non-access to the height of the tree on-the-fly, we decided to experiment with this very simple approach. These branch-and-cut algorithms also uses Gurobi for the first and second phases, where it is used, respectively, to solve the linear relaxation inside a the cutting plane phase and to execute its embedded branch and cut framework. All these branch and cut variations have these 2 phases constrained by 1800 seconds, resulting in a 3600 seconds of overall time constraint. The only cutting planes generated in all these algorithms is the one described in Chapter 10. Gurobi is not allowed to generate any other type of cuts during the second phase. Gurobi agressive pre-solver is allowed too. Gurobi is constrained to use only one core too. The cuts used here are only of the clique consistency cuts type, described in the Chapter 10.

Branch and Price setup

Experimentation 8 is about the branch-and-price algorithm described in Chapter 9, maintaining the same computational constraints and time limits as the other experimental configurations.

The experimental progression establishes baseline performance through Gurobi's branch-and-cut (Experiment 1), evaluates six representative-based variants with different cutting strategies (Experiments 2-7), and concludes with our complete branch-and-price framework (Experiment 8).

11.4 Graph Transformation Analysis

Table D.1 describes the resulting data from the Graph transformation applied on our instance's testbed. The first column remains for the name of the instances. They already condense in the format vX_dY the number of vertices of the original graph as X and its density as Y. Following we have the last column related to the original graph, $|E|$, the number of edges. The remaining columns are related to the output of the graph transformation. The column labeled as $|\overline{V}_0|$ is the number of vertices of the raw graph just after the transformation, which we should expect to be equal to the number of edges in the original graph. Column $|\overline{V}_{\text{iso}}|$ means the number of isolated vertices in the raw transformed graph. It is easy to understand that these isolated vertices are optimal vertex-represented cliques. They are extracted from the transformed graph, reducing its number of vertices, which consequently gives to us the following column, $|\overline{V}|$, which represents the number of the remaining vertices. The input graph our algorithms receive is defined as $K(G) = (\overline{V}, \overline{E})$, which is based on the remaining vertices after subtraction of the isolated vertices. The next column is precisely $|\overline{E}|$, which stands for the number of the edges of the graph $K(G)$. Subsequently, we have the density $\overline{d}$ also based on the input graph $K(G)$. Following the guidelines from Chapter 9, column $|\mathcal{C}|$ is the number of maximal cliques of the graph $K(G)$, which is the same as in the original graph. Finally we have the column presenting the Time in seconds that our transformation procedure spent doing its task.

Table D.1 presents the structural properties of transformed graphs constructed from random graphs used as our benchmark. The transformation exhibits favorable computational properties for sparse instances, with 50-90% of vertices becoming isolated at densities $d \leq 10$, while the graph density remains below 5% for $d \leq 30$. However, a critical phase transition occurs at $d \approx 50$, where graph densities increase sharply to 25-28%, causing exponential growth in maximal cliques from thousands at $d = 30$ to millions at $d = 70$, with corresponding computational times increasing from milliseconds to 94 seconds for the largest instance. This density-dependent complexity suggests that every approach we have to experiment will have its computing tractability delimited to sparse-to-medium density instances, where the sizes of $K(G)$ (fewer than 2,000 non-isolated vertices for $d \leq 30$) appear to be more manageable.

Throughout the remainder of this chapter, we present instance references using an extended notation that includes both original graph properties and the characteristics of the resulting graph transformation. Each instance follows the format `vX_dY (cg: vA, eB, dC%)`, where `vX_dY` represents the original graph with X vertices and Y% density, while the parenthetical `(cg: vA, eB, dC%)` indicates the prop-

erties of the transformed graph with A vertices, B edges, and C% density.

11.5 Baseline Performance: BKG-1

11.5.1 Table description

Table D.2 presents the performance of the baseline method BKG-1 across all test instances. The columns report the following metrics: *Instance* identifies the test case using the extended notation format; $\overline{v}(\mathcal{R}_5)$ represents the best upper bound (incumbent solution) found by BKG-1; $\underline{v}^+(\mathcal{R}_5)$ indicates the best lower bound obtained through the branch-and-bound enumeration process; *gap* shows the optimality gap calculated as $((\overline{v} - \underline{v}^+)/\underline{v}^+) \times 100$; *Nodes* reports the number of enumeration tree nodes explored; *Time(s)* presents the computational time in CPU seconds; $\overline{v}(\mathcal{R}_1)$ denotes the best upper bound achieved by the BKG-1 baseline algorithm in the context of the original graph; and $\underline{v}^+(\mathcal{R}_1)$ represents the best lower bound obtained by BKG-1 through its enumeration process also in the context of the original graph. A dash (-) in the Time column indicates that BKG-1 reached the imposed time limit without solving the instance to optimality. The tilde symbol ($\sim$) appears for instances where BKG-1 could not process due to computational limitations such as memory constraints or excessive model size. Bold values indicate instances solved to proven optimality within the time limit.

11.5.2 Evaluation

The performance analysis of BKG-1 demonstrates its effectiveness as a baseline algorithm for the benchmark instances. Table D.2 presents results showing BKG-1's solution quality across varying instance sizes and densities. BKG-1 achieves provably optimal solutions (indicated by **gap = 0** in the table) for **24 out of 41 benchmark instances** (58.54%). While these instances are classified by their original graph properties in the naming convention, it is important to note that BKG-1 operates on the transformed graphs whose characteristics are detailed in Table D.1. The optimal solutions are concentrated among instances that produce smaller and sparser transformed graphs: all v30_d* instances (which generate graphs $K(G)$ with 6–305 vertices), all v50_d* instances with $d \leq 50$ (producing graphs $K(G)$ with 6–613 vertices), all v70_d* instances with $d \leq 30$ (resulting in graphs $K(G)$ with 12–722 vertices), all v100_d* instances with $d \leq 20$ (generating graphs $K(G)$ with 62–977 vertices), and all v200_d* instances with $d \leq 10$ (producing graphs $K(G)$ with 407–1,698 vertices). This distribution reveals a clear computational complexity boundary where optimal solutions become increasingly difficult to obtain as the transformed graph size and density increase, rather than simply the original graph properties.

The benchmark results clearly demonstrate the escalation of problem difficulty across instance classes based on their transformed graph characteristics. Table D.2 reveals three distinct difficulty regimes when viewed through the scope of these graphs. First, a *tractable regime* corresponding to instances that produce small, sparse graphs $K(G)$ (such as v30_d30 with 125 vertices and 4.22% density, or v70_d20 with 457 vertices and 1.50% density) where BKG-1 consistently achieves optimal solutions with minimal computational effort, solution times typically under 100 seconds, and negligible optimality gaps. Second, a *challenging regime* where graphs $K(G)$ exhibit moderate size and density (such as v100_d30 with 1,485 vertices and 2.00% density, or v150_d20 with 2,231 vertices and 0.69% density) where optimal solutions become sporadic, with increasing gaps between upper and lower bounds ranging from 2.8% to 300.0%, computational times extending to the time limit, and the number of processed nodes varying dramatically from single digits to over 100,000. Third, a *hard regime* where graphs $K(G)$ become large and dense (such as v200_d50 with 9,950 vertices and 7.16% density) where optimal solutions are rare or unattainable within reasonable computational limits, large optimality gaps emerge (up to 100% for `v200_d50`), and some instances exceed the computational capacity (indicated by $\sim$ symbols for high-density instances producing transformed graphs with millions of edges).

The performance characteristics of the algorithm reveal important insights about the relationship between original graph G structure and graph $K(G)$ complexity. For sparse original graphs ($d \leq 10$), which produce small or very sparse graphs $K(G)$ (see Table D.1), BKG-1 achieves optimality for all tested cases up to $|V| = 200$. The node processing statistics demonstrate the algorithm's efficiency on instances producing manageable transformed graphs $K(G)$, with many requiring zero branching nodes (solved at the root), while instances generating larger, denser graphs $K(G)$ show significant variability, with some of them terminating after exploring just one node due to difficulty in improving bounds, while others process over 100,000 nodes before reaching the time limit. The time-to-optimality for solved instances ranges from milliseconds for instances producing small graphs $K(G)$ to several hundred seconds for those generating moderate-sized graphs $K(G)$ like `v50_d50` (cg: v613, e20001, d10.66%), which required 412.194 CPU seconds with 57,651 nodes processed.

BKG-1 serves as an effective baseline for algorithm evaluation due to its single-threaded consistency, competitive performance achieving optimal solutions on 24 out of 41 instances, and ability to generate meaningful bounds across the full problem spectrum. As a well-established implementation using standard techniques, it provides a reliable reference point for comparing algorithmic innovations.

11.6 Selection of the Representative-Based Algorithm Configuration

11.6.1 Description of the Tables

The representative-based algorithm evaluation employs six pairs of tables corresponding to experiments 2–7. Each pair consists of a solution quality table and a branch-and-cut statistics table that together provide the performance analysis for each experimental configuration.

Solution Quality Tables. Each experiment's first table (e.g., Table D.4 for VREP-2) presents solution quality metrics with the following columns: *Instance* using original graph notation; $\underline{v}(\mathcal{R}_{10})$ (lower bound based on the initial formulation without cuts); $\overline{v}(\mathcal{R}_{10})$ (best upper bound for the final formulation); $\underline{v}(\mathcal{R}_{10}^{+})$ (final lower bound after adding clique consistency cuts); *Gap* (optimality gap); *Time(s)* (computational time in CPU seconds); $\overline{v}(\mathcal{R}_1)$ (upper bound mapped to original graph); and $\underline{v}^{+}(\mathcal{R}_1)$ (lower bound mapped to original graph). A dash (-) indicates time limit reached, tilde ($\sim$) indicates computational failure, and bold values indicate proven optimality.

Branch-and-Cut Statistics Tables. Each experiment's second table (e.g., Table D.5 for VREP-2) reports algorithmic execution details with columns: *Instance* using original graph notation; *Nodes* (number of explored enumeration nodes); *Root Cuts* (clique consistency cuts added during the root cutting plane phase); and *Node Cuts* (clique consistency cuts added during tree enumeration). The tilde symbol ($\sim$) indicates computational intractability due to memory limitations or excessive model size.

Model Dimension Table. Additionally, Table D.3 presents the size characteristics of the initial representative-based formulation loaded into the Gurobi solver. The columns report: *Instance* using original graph notation; *Rows* (number of constraint rows); *Columns* (number of decision variables); and *Non-zeros* (number of non-zero coefficients in the constraint matrix).

11.6.2 Evaluation

A critical characteristic of the representative-based formulation is its tendency to generate extremely large mathematical models, particularly for graphs $K(G)$ with many vertices or high edge density. As shown in Table D.3, the formulation size grows dramatically with these properties. For instance, v50_d70 (cg: v858, e103218, d28.07%) generates models with over 32 million non-zero coefficients, while v70_d70 (cg: v1691, e389049, d27.23%) exceeds 349 million non-zeros. Instances generating

models with more than one million non-zero coefficients prove computationally intractable in our experimental environment, with the sole exception of v50_d50 (cg: v613, e20001, d10.66%), which some configurations manage to process albeit with poor solution quality exceeding 22% optimality gap.

The primary evaluation metric focuses on solution quality, measured by the number of instances solved to optimality and the quality of non-optimal solutions. The computational tractability shows strong correlation with both the size and density of the transformed graph. Small graphs $K(G)$ such as v30_d10 (cg: v6, e6, d40%) and v50_d5 (cg: v6, e6, d40%) solve instantly across all configurations despite their relatively high density. As the graph $K(G)$ size increases, performance degrades rapidly. Instance v30_d70 (cg: v305, e13530, d29.18%), with moderate size but high density, already presents challenges with gaps ranging from 12.5% to 26.09% across different configurations.

The best-cuts strategy (Experiment 2) achieves 20 optimal solutions but exhibits weakness in lower bound quality for unsolved instances. For v100_d30 (cg: v1485, e22074, d2.00%), a medium graph $K(G)$, this approach generates relatively few cuts, leading to poor bound quality. Conversely, the all-cuts strategy generates substantially larger formulations, as evidenced by v30_d70 (cg: v305, e13530, d29.18%) requiring 143,258 root cuts in Experiment 4, overwhelming the solver despite the moderate graph $K(G)$ size.

The best-cuts-per-representative strategy (Experiments 3 and 6) emerges as a balanced approach. This strategy maintains moderate formulation sizes while achieving competitive solution quality, solving 20-21 instances to optimality. For instance v30_d70 (cg: v305, e13530, d29.18%), this approach requires only 427-644 root cuts compared to 143,258 in the all-cuts strategy, yet achieves similar optimality gaps. The strategy proves particularly effective for graphs $K(G)$ with low to moderate density, such as v70_d30 (cg: v722, e6735, d2.59%), where it achieves gaps below 1.17%.

The incorporation of node cuts during enumeration yields mixed results. For graphs $K(G)$ like v70_d30 (cg: v722, e6735, d2.59%), additional node cuts contribute between 2,000 and 4,500 constraints, mildly increasing its LB. The most successful configuration combines node cuts with the best-cuts-per-representative strategy (Experiment 6), which achieves one additional optimal solution for v120_d20 (cg: v1416, e8214, d0.82%), a large sparse graph $K(G)$, suggesting marginal benefits that may not justify the additional computational cost.

At the root node, all branch-and-cut variants demonstrate comparable performance, with 15-16 instances solved optimally through the cutting plane phase alone. This consistency indicates that the initial strengthening achieved through separation heuristics effectively tightens the formulation for smaller graphs $K(G)$ and

those with favorable structural properties, regardless of the specific cutting strategy employed.

The experimental results reveal clear patterns based on transformed graph properties. Small graphs $K(G)$ (fewer than 100 vertices) generally prove solvable regardless of density, as seen with v30_d30 (cg: v125, e327, d4.22%) and v50_d10 (cg: v42, e60, d6.97%). Moderately-sized graphs $K(G)$ with low density remain tractable, such as v100_d20 (cg: v977, e4857, d1.02%) and v70_d20 (cg: v457, e1563, d1.50%). However, the combination of large size and moderate-to-high density creates intractability.

The representative-based formulation provides valuable insights into VCCP complexity while demonstrating both strengths and limitations. The formulation successfully solves numerous benchmark instances, particularly those with small or large sparse graphs $K(G)$. The rapid growth in model size becomes prohibitive for large, dense graphs $K(G)$. For instance, v150_d70 (cg: v7823, e7783854, d25.44%) and v200_d70 (cg: v13930, e24489456, d25.24%) generate models far exceeding available computational resources, with the latter containing nearly 25 million edges in its graph $K(G)$. The best-cuts-per-representative strategy offers the most promising balance between formulation strength and computational tractability, though even this approach struggles when transformed graphs exhibit both large vertex counts (exceeding 2,000) and moderate-to-high density (above 10%).

The best-cuts-per-representative approach achieves a good balance between cut generation and computational efficiency. While Experiments 3 and 6 both solve several instances to optimality, Experiment 6 additionally solves v120_d20 (cg: v1416, e8214, d0.82%), making it marginally superior under our solution quality metric. This configuration therefore serves as the representative-based formulation benchmark for comparison with the Branch-and-Price algorithm in the following section.

11.7 Branch-and-Price Framework Evaluation

11.7.1 Tables and Figure Description

The experimental results for the Branch-and-Price algorithm are presented across four tables and one comprehensive figure. Throughout these tables, specific notation conventions are used consistently. The tilde symbol ($\sim$) indicates missing data for instances that could not be processed. A dash (–) appears in two contexts: in the $\underline{v}(\mathcal{R}_5)$ column, it indicates that CGA did not terminate with an optimal integral solution; in the Time column, it indicates that the algorithm reached the imposed time limit. Values marked with an asterisk (*) indicate that CGA at root node terminated prematurely before proving that no columns with negative

reduced cost exist, meaning these values do not represent valid lower bounds for the formulation 9.7. Bold values indicate instances solved to proven optimality.

Table D.16 presents *CGA results at the root node* of the Branch-and-Price algorithm. The columns report: $\overline{v}$(CGM) (upper bound from the initial heuristic solution), $\underline{v}(\mathcal{R}_6)$ (lower bound from the initial restricted formulation), $\underline{v}(\mathcal{R}_6^+)$ (lower bound after column generation at the root node), Root Time (computational time in CPU seconds at the root node), Generated Columns (number of columns added during column generation), Col Gen Iterations (number of column generation iterations performed), and $\overline{\underline{v}}(\mathcal{R}_5)$ (integer solution value when column generation terminates with an integral solution, shown in bold when optimal).

Table D.17 presents *the performance metrics* for the Branch-and-Price algorithm. The columns report: $\overline{v}(\mathcal{R}_6)$ (best upper bound found by the algorithm), $\underline{v}^+(\mathcal{R}_6^+)$ (best lower bound obtained through column generation and branching), Gap (optimality gap calculated as $((\overline{v} - \underline{v}^+)/\underline{v}^+) \times 100$), Nodes (number of nodes explored in the enumeration tree), and Time(s) (total computational time in CPU seconds).

Table D.18 provides detailed *statistics about the enumeration tree structure* explored by the Branch-and-Price algorithm. The columns report: Deepest Depth (maximum depth reached in the search tree), Average Depth (mean depth of all explored nodes), Branching Count (total number of branching operations performed), and three pruning categories indicating how nodes were eliminated from further exploration—by bound (when the lower bound exceeded the incumbent solution), by integrality (when an integer solution was found), and by infeasibility (when no feasible solution existed for that branch).

Table D.19 validates the solution feasibility of the Branch-and-Price algorithm by verifying that solutions obtained for the transformed VCCP correctly map back to feasible solutions for the original ECCP. The columns show: VCCP Solution Feasibility (confirming that the solution satisfies all vertex coverage constraints in the transformed graph), ECCP Solution Feasibility (verifying that the mapped solution covers all edges in the original graph), and $\overline{v}(\mathcal{R}_1)$ (the objective value of the best solution found, with bold values indicating proven optimal solutions).

Figure 11.1 provides a visual comparison of the final upper and lower bounds achieved across the three algorithms composing the third block of experimentation: BKG-1, VREP-6 and the proposed Branch-and-Price (BP). The main shape is a circle, where light gray rays indicate a zone corresponding to the instance whose name is in the same direction, arranged radially in the outer region of the circle. Although this zone is not delimited, the proximity of the algorithm plots to the light gray ray indicates their correspondence to their instance target. The reader may think about a gray ray as an axis starting from zero, represented by the center of the circle, going to infinity. The border of the circle delimits a position in this

ray that is used as a reference for all the other data. The border is draw in black and depicts the value of the best upper bound found by BKG-1 for the ray related instance. BKG-1 lower bound is presented as a light gray arc, that crosses the ray precisely at the ratio $(\mathrm{LB}_{\text{bkg-1}}/\mathrm{UB}_{\text{bkg-1}}) \leq 1$, which is always in the inner region of the circle, by definition. Note that this arc only loosely delimits the instance zone we discussed before. Now, it is intuitively easy to see the BKG-1 bound's interval. The other two algorithms, VREP-6 and BP, have their bound's intervals plotted in each side, near to the ray. VREP-6 bound's interval is represented by a double-arrow segment. The arrow pointing to the center of the circle delimits its lower bound, and the other arrow its upper bound. For the BP bound's interval, we display it as a segment with T-bars at its ends, also delimiting the lower and upper bounds found by the algorithm. The bar near to the center represents its lower bound and the other, its upper bound. All the values of the bounds share the same proportional relation with BKG-1 upper bound as discussed before. Interesting patterns can be seen when an optimal solution is found. If BKG-1 attains such a solution, this means the light gray arc collapsing into the border, which turns the border to be colored with its light gray. If VREP-6 found an optimal solution, then a shape that looks like an hourglass is formed. In the case of an optimal solution being found by BP, the bars collapse into a small black dash, drawn perpendicular to the ray at the point of intersection, the center of the dash. When some data is missing, the reader can easily see the lack of the plot related to the algorithm's data. If an interval is plotted entirely inside the circle, it means its related upper bound is better than BKG-1. This figure condenses all the complex information of several tables in an intuitive way to compare between the performance of the competing algorithms.

11.7.2 Evaluation

The branch-and-price algorithm demonstrates a distinct performance profile compared to the baseline algorithms BKG-1 and VREP-6, achieving optimality in 20 out of 41 test instances (48.8%). This success rate falls short of BKG-1's performance with 24 optimal solutions (58.5%) and VREP-6's achievement of 21 optimal solutions (51.2%). However, as illustrated in Figure 11.1, the comparative analysis reveals nuanced trade-offs that highlight the complementary strengths and limitations of each algorithmic approach across the solution quality spectrum.

The radar chart visualization in Figure 11.1 reveals a fundamental performance dichotomy: while BKG-1 consistently provides superior lower bounds across virtually all instances, branch-and-price maintains computational feasibility in scenarios where BKG-1 fails entirely. The most striking observation from the figure is the systematic dominance of BKG-1's lower bounds (light gray arcs) over branch-and-price

bounds (T-bars), particularly evident in medium-density instances such as v70_-d30 (cg: v722, e6735, d2.59%), v100_d30 (cg: v1485, e22074, d2.00%), v120_d20 (cg: v1416, e8214, d0.82%), and v150_d20 (cg: v2231, e17073, d0.69%). This lower bound inferiority represents a significant theoretical disadvantage for branch-and-price, as tighter relaxations typically correlate with more efficient enumeration procedures. However, the enumeration statistics reveal a complex and instance-dependent relationship between the algorithm's search efficiencies. While v70_d30 (cg: v722, e6735, d2.59%) demonstrates BKG-1's superior enumeration with 1,111 nodes in 3 seconds compared to branch-and-price's 4,071 nodes before timeout, the pattern reverses dramatically in other instances. For v100_d30 (cg: v1485, e22074, d2.00%), BKG-1 explores over 111,000 nodes before reaching the time limit while branch-and-price examines only 2,015 nodes, representing a 50-fold efficiency advantage. Similarly, v150_d20 (cg: v2231, e17073, d0.69%) shows BKG-1 exploring 123,732 nodes versus branch-and-price's 2,491 nodes. This disparity suggests that despite producing weaker bounds, branch-and-price employs a fundamentally different and possibly more efficient enumeration strategy that can navigate the solution space with dramatically fewer node evaluations in most medium-to-large instances.

However, the figure also highlights branch-and-price's critical advantage in computational robustness at specific density thresholds. While both algorithms fail on the highest density instances (v120_d70 (cg: v4998, e3235839, d25.91%), v150_d70 (cg: v7823, e7783854, d25.44%), v200_d70 (cg: v13930, e24489456, d25.24%)), branch-and-price demonstrates superior performance in the intermediate high-density region (d50 instances). For v120_d50 (cg: v3570, e493824, d7.75%), v150_d50 (cg: v5588, e1183398, d7.58%), and v200_d50 (cg: v9950, e3546237, d7.16%), branch-and-price provides substantially better upper bounds than BKG-1 while maintaining reasonable lower bounds. The most striking example occurs in v200_d50 (cg: v9950, e3546237, d7.16%), where BKG-1 fails catastrophically with an extremely high upper bound of 2010 and a lower bound of 0, while branch-and-price delivers reasonable bounds of 800 (upper) and 469 (lower), representing the only viable approach for obtaining meaningful mathematical bounds on this challenging instance.

An expected behavior of the branch-and-price algorithm lies in its root node performance, where 16 of the 20 optimal solutions are achieved without requiring any branching. Among these root-optimal instances, eight solve immediately with zero column generation, while the remaining eight require modest column generation efforts with at most 170 columns and 8 iterations. These results are expected because BKG-1 also reaches them without requiring any branching, which indicates the linear relaxation is allowed to reach optimal solutions in case of having the optimal columns at hand in the formulation. Thus, for some instances, CGA provided them

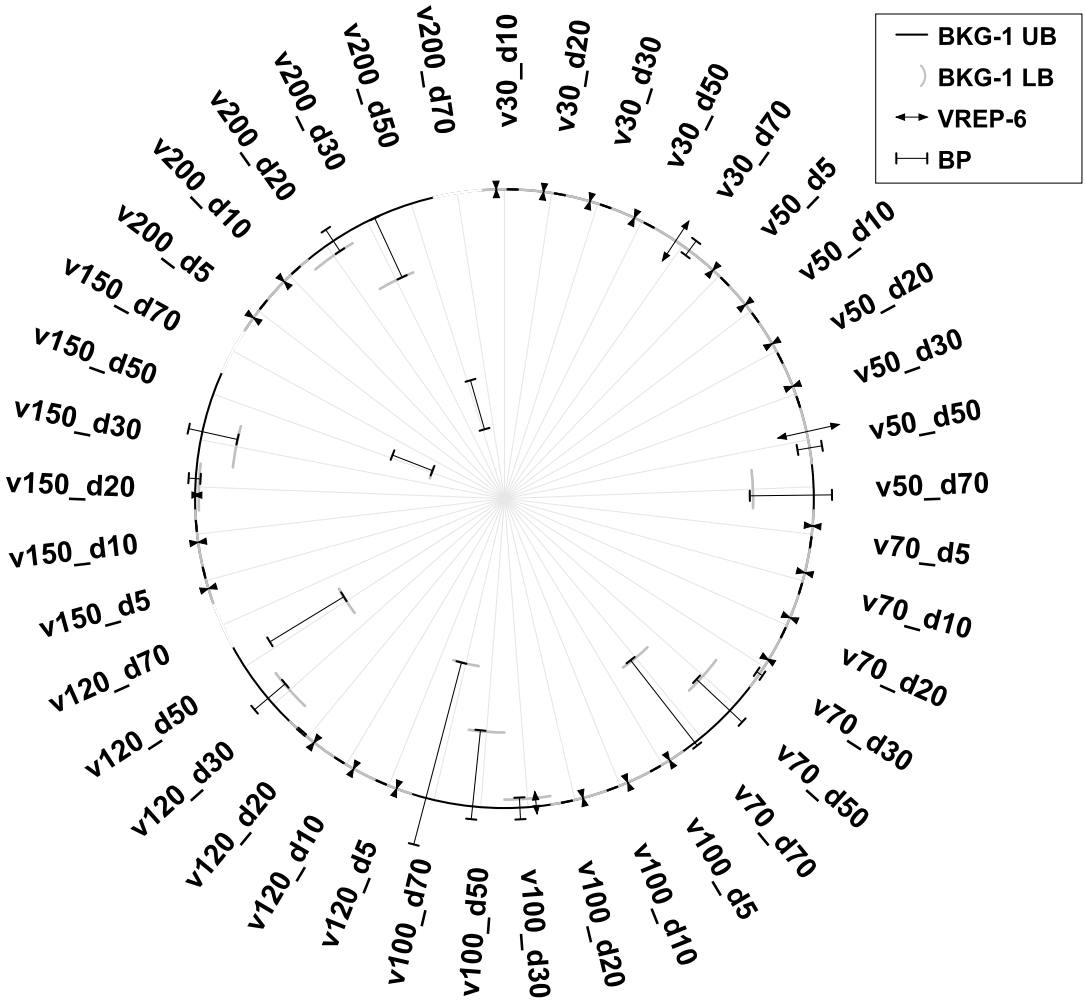

Figure 11.1: Part II results.

to BP. The instances achieving root-node optimality, clearly visible in Figure 11.1 as standalone bars indicating that branch-and-price bounds collapsed into optimal values, cluster primarily in the small graphs/low-density regions of the chart.

The tractable region for branch-and-price can be characterized by instances with optimality gaps below 10%, which correspond to very small graphs $K(G)$ ($|\overline{V}| \leq 100$) regardless of density, medium-sized instances ($100 < |\overline{V}| < 1{,}000$) with low average degree, and larger sparse instances with extremely low density. The reader can notice this region in Figure 11.1 by verifying the instances where BP presents its results near to the border of the circle, with small optimality gap intervals (segments with T-bars). The performance degradation becomes apparent for instances where BP presents increasing solution quality gap, due to the dramatically growth of computational requirements.

The branching tree structure analysis, while not directly visible in Figure 11.1, correlates with the error bar patterns observed across instances. Tree depths rarely

exceed 40 levels, with average depths clustering around half the maximum depth. The consistent absence of integrality-based pruning throughout all instances suggests that fractional solutions persist deep into the enumeration tree, contributing to the substantial optimality gaps observed in the radar chart.

The lower bound quality achieved through column generation, while consistently inferior to BKG-1 as demonstrated throughout Figure 11.1, provides value even when optimality cannot be proven within time limits. These bounds offer theoretical benchmarks for solution quality assessment and enable practitioners to quantify the optimality gap of heuristic solutions. In scenarios where BKG-1 becomes computationally intractable, branch-and-price has the potential to be the only viable approach for obtaining meaningful lower bounds, despite the substantial computational investment required.

The algorithm's performance profile ultimately reflects the inherent complexity of the Edge Clique Cover Problem and the challenges associated with exact solution approaches. While Figure 11.1 clearly demonstrates that branch-and-price fails to dominate existing methods in bound quality across any instance class, it provides a theoretically sound framework that extends the computational frontier beyond the reach of conventional approaches. The ability to maintain feasibility on the most challenging instances, combined with rigorous optimality guarantees within the tractable region, establishes branch-and-price as a valuable addition to the algorithmic toolkit for clique cover problems.

Chapter 12

Conclusions and Future Work

This thesis investigated Set Covering Problem formulations for the Edge Clique Cover Problem, developing both heuristic frameworks and exact solution methods. Our aim was not to offer fine-tuned algorithms for specific formulations, but rather to explore the fundamental challenges that come with SCP-based approaches to ECCP.

The computational experiments reveal that clique sampling strategies can generate restricted formulations that significantly outperform traditional ECCP heuristics. Existing ECCP heuristics, when used in isolation, prove inadequate for challenging instances, as demonstrated by CGM-X consistently lagging behind sampling-based frameworks that incorporate the same heuristic as a component within sophisticated restricted formulations.

The graph transformation provides a fundamentally different perspective that leverages simpler graph structures, enabling theoretically sound exact algorithms. The branch-and-price approach demonstrates computational feasibility on instances where traditional methods fail, though this appears primarily due to formulation size management rather than algorithmic superiority. The representative-based VCCP formulation, while providing theoretical insights, encounters prohibitive model growth that represents a fundamental barrier limiting practical applicability.

The experimental evaluation reveals three distinct performance regimes: sparse instances remain tractable for sophisticated approaches, moderate densities create challenging instances requiring careful algorithmic design, and dense instances present fundamental computational barriers. Beyond certain complexity thresholds, both commercial solvers and branch-and-price encounter enumeration trees so vast that meaningful pruning opportunities vanish, leading to unproductive exploration where linear relaxation guidance becomes insufficient.

Future research should explore fundamentally different algorithmic paradigms that sidestep the enumeration explosion limiting current approaches. The most nat-

ural evolution involves integrating sampling-based techniques with exact branch-and-price algorithms, as the dynamic column accumulation creates increasingly attractive restricted formulations that LS-RCSH could exploit effectively. Given the apparent approach to practical limits for these NP-hard problems, specialized algorithms designed for distributed computing systems represent another promising direction, leveraging the inherent parallelism in column generation procedures.

Alternative research directions include exploring hybrid strategies combining column generation insights with primal solution methods designed for large-scale formulations, such as Lagrangian relaxation techniques. Rather than pursuing incremental improvements to existing exact frameworks, investigating specific structural properties of graph classes that maintain tractability where general-purpose methods fail may yield greater returns. The graph transformation approach could also be extended to related optimization problems involving clique structures, enabling application of proven algorithmic techniques from related domains.

References

ORLIN, J. "Contentment in graph theory: covering graphs with cliques". In: *Indagationes Mathematicae (Proceedings)*, v. 80, pp. 406–424. Elsevier, 1977.

KOU, L. T., STOCKMEYER, L. J., WONG, C.-K. "Covering edges by cliques with regard to keyword conflicts and intersection graphs", *Communications of the ACM*, v. 21, n. 2, pp. 135–139, 1978.

LUND, C., YANNAKAKIS, M. "On the hardness of approximating minimization problems", *Journal of the ACM (JACM)*, v. 41, n. 5, pp. 960–981, 1994.

AGARWAL, P. K., ALON, N., ARONOV, B., et al. "Can visibility graphs be represented compactly?" *Discrete & Computational Geometry*, v. 12, n. 3, pp. 347–365, 1994.

RAJAGOPALAN, S., VACHHARAJANI, M., MALIK, S. "Handling irregular ILP within conventional VLIW schedulers using artificial resource constraints". In: *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2000)*, pp. 157–164, 2000.

CONTE, A., GROSSI, R., MARINO, A. "Clique covering of large real-world networks". In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pp. 1134–1139, 2016.

BLANCHETTE, M., KIM, E., VETTA, A. "Clique cover on sparse networks". In: *2012 Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pp. 93–102. SIAM, 2012.

GRAMM, J., GUO, J., HÜFFNER, F., et al. "Data reduction, exact, and heuristic algorithms for clique cover". In: *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, pp. 86–94. Society for Industrial and Applied Mathematics, 2006.

HEVIA, A., KALLUS, B., MCCLINTIC, S., et al. "Solving Edge Clique Cover Exactly via Synergistic Data Reduction", *arXiv preprint arXiv:2306.17804*, 2023.

DESROSIERS, J., LÜBBECKE, M. E. "Branch-Price-and-Cut Algorithms". In: *Wiley Encyclopedia of Operations Research and Management Science*, John Wiley & Sons, Ltd, 2011.

LÜBBECKE, M. E., DESROSIERS, J. "Selected topics in column generation", *Operations research*, v. 53, n. 6, pp. 1007–1023, 2005.

PICCIANI, D., LUCENA, A. "Conceptual clique sampling frameworks to design solution algorithms for the Edge Clique Cover Problem", *RAIRO-Operations Research*, v. 59, n. 1, pp. 523–547, 2025.

JOHNSON, D. S., GAREY, M. R. *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman, 1979.

GAVRIL, F. "Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph", *SIAM Journal on Computing*, v. 1, n. 2, pp. 180–187, 1972.

BOOLE, G. "Of Propositions Numerically Definite, 1868", *Chapter IV of Studies in Logic and Probability, Waters, London*, 1952.

PIEPHO, H.-P. "An algorithm for a letter-based representation of all-pairwise comparisons", *Journal of Computational and Graphical Statistics*, v. 13, n. 2, pp. 456–466, 2004.

GRAMM, J., GUO, J., HÜFFNER, F., et al. "Algorithms for compact letter displays: Comparison and evaluation", *Computational Statistics & Data Analysis*, v. 52, n. 2, pp. 725–736, 2007.

MARKHAM, A., GROSSE-WENTRUP, M. "Measurement Dependence Inducing Latent Causal Models". In: *Conference on Uncertainty in Artificial Intelligence*, pp. 590–599. PMLR, 2020.

GUILLAUME, J.-L., LATAPY, M. "Bipartite structure of all complex networks", *Information processing letters*, v. 90, n. 5, pp. 215–221, 2004.

ROBERTS, F. S. "Applications of edge coverings by cliques", *Discrete applied mathematics*, v. 10, n. 1, pp. 93–109, 1985.

ERDÖS, P., GOODMAN, A. W., PÓSA, L. "The representation of a graph by set intersections", *Canadian Journal of Mathematics*, v. 18, pp. 106–112, 1966.

MCKEE, T. A., MCMORRIS, F. R. *Topics in intersection graph theory*. SIAM, 1999.

HERMELIN, D., RIZZI, R., VIALETTE, S. "Algorithmic aspects of the intersection and overlap numbers of a graph". In: *International Symposium on Algorithms and Computation*, pp. 465–474. Springer, 2012.

ABDULLAH, W. M., HOSSAIN, S. "A sparse matrix approach for covering large complex networks by cliques". In: *International Conference on Computational Science*, pp. 505–517. Springer, 2022.

BORNDÖFER, R. *Aspects of Set Packing, Partitioning, and Covering An Analysis of Example*. Phd thesis, Technischen Universität Berlin, 1998.

MEHROTRA, A., TRICK, M. A. "A column generation approach for graph coloring", *informs Journal on Computing*, v. 8, n. 4, pp. 344–354, 1996.

STÜTZLE, T. *Local search algorithms for combinatorial problems*. Phd thesis, Darmstadt University of Technology, 1998.

TOMITA, E., TANAKA, A., TAKAHASHI, H. "The worst-case time complexity for generating all maximal cliques and computational experiments", *Theoretical computer science*, v. 363, n. 1, pp. 28–42, 2006.

LUCENA, A., MACULAN, N., SIMONETTI, L. "Reformulations and solution algorithms for the maximum leaf spanning tree problem", *Computational Management Science*, v. 7, n. 3, pp. 289–311, 2010.

PARK, K., LEE, K., PARK, S. "An extended formulation approach to the edge-weighted maximal clique problem", *European Journal of Operational Research*, v. 95, n. 3, pp. 671–682, 1996.

HOSSEINIAN, S., FONTES, D. B., BUTENKO, S., et al. "The maximum edge weight clique problem: formulations and solution approaches". In: *Optimization Methods and Applications*, Springer, pp. 217–237, 2017.

SABÓIA, C. H. M. *Um Algoritmo Branch-and-Price para Instâncias de Grande Porte do Modelo Brasileiro de Planejamento da Expansão da Geração Elétrica a Longo Prazo*. Tese de Doutorado, Programa de Engenharia de Sistemas e Computação - PESC/COPPE, Universidade Federal do Rio de Janeiro, 2013.

CHANG, G. J. "The domatic number problem", *Discrete Mathematics*, v. 125, n. 1, pp. 115–122, 1994.

CAMPÊLO, M. B., CAMPOS, V. A., CORRÊA, R. C. "On the asymmetric representatives formulation for the vertex coloring problem", *Electron. Notes Discret. Math.*, v. 19, pp. 337–343, 2005.

DANNA, E., ROTHBERG, E., PAPE, C. L. "Exploring relaxation induced neighborhoods to improve MIP solutions", *Mathematical Programming*, v. 102, n. 1, pp. 71–90, 2005.

YÜCEOĞLU, B., ŞAHIN, G., VAN HOESEL, S. P. "A column generation based algorithm for the robust graph coloring problem", *Discrete Applied Mathematics*, v. 217, pp. 340–352, 2017.

GONÇALVES, J. F., RESENDE, M. G., TOSO, R. F. "Biased and unbiased random-key genetic algorithms: An experimental analysis", *AT&T Labs Research, Florham Park*, 2012.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., et al. *Introduction to algorithms*. MIT press, 2022.

Appendix A

Set Covering Problem Heuristic Implementation

A.1 Problem Definition

Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a collection of n finite sets, and let $U = \bigcup_{j=1}^{n} P_j = \{1, 2, \ldots, m\}$ be the universe of elements to be covered. Each set $P_j \in \mathcal{P}$ has an associated cost $c_j > 0$. A subset $\mathcal{S} \subseteq \mathcal{P}$ is called a *cover* if $\bigcup_{P_j \in \mathcal{S}} P_j = U$. The Set Covering Problem (SCP) seeks to find a cover $\mathcal{S}^*$ that minimizes the total cost:

$$\min_{\mathcal{S} \subseteq \mathcal{P}} \left\{ \sum_{P_j \in \mathcal{S}} c_j : \bigcup_{P_j \in \mathcal{S}} P_j = U \right\} \tag{A.1}$$

To contextualize this heuristic's application, Chapter 9 presents our branch-and-price algorithm for the ECCP via a graph transformation that enables a reformulation from ECCP to VCCP. The SCP which we solve is based on this VCCP reformulation. The sets P_j correspond to vertex-represented cliques (columns of the formulation), and the universe U represents the vertices of the transformed graph, which correspond to the edges of the original graph that must be covered. Each clique can cover multiple vertices, and we seek a minimum-cost collection of cliques that covers all vertices of the transformed graph. From here, the reader may consider the SCP as a reinterpretation of the VCCP.

A.2 Integration with Linear Relaxation

At each node of the branch-and-price tree, after solving the linear relaxation through column generation, we may obtain fractional solution values $x_j^* \in [0, 1]$ for each column j in the restricted master problem. To convert this fractional solution into a feasible integer solution (providing a primal bound), we use the linear relaxation

solution to guide the greedy selection process. Rather than treating all columns equally as in a unit-cost SCP, we derive modified costs from the LP solution:

$$c_j = 1 - x_j^* \tag{A.2}$$

This transformation has the following interpretation:

- Columns with high fractional values ($x_j^* \approx 1$) receive low costs, making them more attractive to the greedy algorithm.
- Columns with low fractional values ($x_j^* \approx 0$) receive high costs, discouraging their selection.

A.3 Two-Phase Heuristic Algorithm

The heuristic consists of two phases: (i) a greedy construction phase that builds an initial solution using the LP-informed costs, and (ii) an improvement phase that removes redundant variables to obtain a minimal solution. The algorithms presented here are based on the works of GONÇALVES *et al.* (2012) and CORMEN *et al.* (2022).

A.3.1 Complete Algorithm

The complete heuristic combines both phases to produce an efficient primal solution, as shown in Algorithm 6:

Algorithm 6 Complete LP-Guided SCP Heuristic

Require: Linear relaxation solution x^*, collection of sets $\mathcal{P}$, universe U
Ensure: Integer feasible solution S_{final}
1: Compute modified costs: $c_j \leftarrow 1 - x_j^*$ for all j
2: $S_{greedy} \leftarrow$ GreedySetCover(U, $\mathcal{P}$, c) {Algorithm 7}
3: $S_{final} \leftarrow$ GreedyUncover(S_{greedy}, $\mathcal{P}$, U) {Algorithm 8}
4: **return** S_{final}

A.3.2 Phase 1: Greedy Set Cover Construction

The greedy construction phase follows the classical greedy algorithm SCP, selecting variables based on their cost-efficiency ratio. The algorithm maintains a max-heap of variable candidates ordered by their cost-to-coverage ratio, as detailed in Algorithm 7.

Algorithm 7 Greedy Set Cover Construction

Require: Vertex universe U, collection of sets $\mathcal{P}$, modified costs c from Eq. (A.2)
Ensure: Solution set $S \subseteq \mathcal{P}$

```
Initialize solution S ← ∅
Initialize covered vertices V_covered ← ∅
Initialize variable candidates with cost-efficiency ratios
Build max-heap H of candidates ordered by cost-efficiency
while not all vertices are covered do
    v_max ← extract maximum cost-efficiency variable from H
    S ← S ∪ {v_max}
    Update V_covered with vertices covered by v_max
    for each vertex u newly covered by v_max do
        for each variable v that covers vertex u do
            if v ∉ S then
                Decrease coverage count of v
                Recompute cost-efficiency of v
                Restore heap property for v in H
            end if
        end for
    end for
end while
return S
```

Cost-Efficiency Computation

The cost-efficiency for each variable candidate is computed as:

$$\text{efficiency}(v) = \begin{cases} -\frac{c_v}{\text{uncovered\_vertices}(v)} & \text{if uncovered\_vertices}(v) > 0 \\ -\infty & \text{if uncovered\_vertices}(v) = 0 \end{cases} \tag{A.3}$$

where $c_v = 1 - x_v^*$ is the modified cost derived from the linear relaxation, and uncovered_vertices(v) is the number of currently uncovered vertices that variable v would cover. The negative sign converts the minimization criterion (minimum cost-per-vertex) into a maximization criterion suitable for the max-heap data structure.

A.3.3 Phase 2: Greedy Uncovering (Solution Improvement)

After the construction phase, the algorithm attempts to improve the solution by removing redundant variables. A variable is considered redundant if all vertices it covers are also covered by other variables in the current solution. This process is formalized in Algorithm 8.

Algorithm 8 Greedy Uncovering

Require: Solution S from Phase 1, collection of sets $\mathcal{P}$, vertices to cover V
Ensure: Improved solution S'

Initialize array vertex_count$[\cdot] \leftarrow 0$
for each variable $v \in S$ **do**
  for each vertex u covered by v **do**
    vertex_count$[u] \leftarrow$ vertex_count$[u] + 1$
  end for
end for
$R \leftarrow \emptyset$ {Set of redundant variables}
for $i \leftarrow |S| - 1$ **down to** 0 **do**
  $v \leftarrow S[i]$
  if IsRedundant$(v,$ vertex_count$)$ **then**
    $R \leftarrow R \cup \{v\}$
    for each vertex u covered by v **do**
      vertex_count$[u] \leftarrow$ vertex_count$[u] - 1$
    end for
  end if
end for
$S' \leftarrow S \setminus R$
return S'

Appendix B

Table of Acronyms

The following table summarizes the acronyms and their definitions used in the context of the presented formulations and heuristics.

Acronym	**Definition**
BKA	Bron-Kerbosch Algorithm
BKG	Bron-Kerbosch-Gurobi algorithm
BKG-1	Single-core version of BKG algorithm used as baseline
BLS	Baseline Solution
BP	Branch-and-Price
CGA	Column Generation Approach
CGM	A heuristic for solving the ECCP, based on the work of Conte, Grossi, and Marino CONTE *et al.* (2016)
CGM-X	An algorithm embedding CGM, which calls it multiple times under a time or iteration limit
CPU	Central Processing Unit
CSH	Clique Sampling Heuristic
DNP	Domatic Number Problem
ECC	Edge Clique Cover
ECCP	Edge Clique Cover Problem
ET-LPR	Enumeration Tree LPR. Smallest LPR bound available at any Gurobi enumeration tree node that is still open
GCC	GNU Compiler Collection
IG	Intersection Graph
IP	Integer Programming
K(G)	Transformed graph constructed from original graph G in chapter 9
LPR	Linear Programming Relaxation
LS	Local Search
LS-RCSH	Local Search Reduced Cost Sampling Heuristic
MEWCP	Maximum Edge Weight Clique Problem
MIP	Mixed Integer Programming
MVWCP	Maximum Vertex Weighted Clique Problem
NP	Nondeterministic Polynomial time complexity class

Acronym	**Definition**
PP	Pricing Problem
RAM	Random Access Memory
RCSH	Reduced Cost Sampling Heuristic
RMP	Restricted Master Problem
R-SCP-ECCP	Restricted SCP-ECCP formulation
R-SCP-VCCP	Restricted SCP-VCCP formulation
SCP	Set Covering Problem
SCP-VCCP	Set Covering Problem formulation for Vertex Clique Cover Problem
SF-SCP-ECCP	Soft Fixing R-SCP-ECCP formulation
SFI	Soft Fixing Inequality
VCP	Vertex Coloring Problem
VCCP	Vertex Clique Cover Problem

Table B.1: Acronyms used in the Thesis.

Appendix C

Tables of Experiments for Part I

			CGM-X			BKG			
Instance	$\|E\|$	$\|\mathcal{C}\|$	**Best**	**# Runs**	**Time(s)/iter**	$\underline{v}(\mathcal{R}_1)$	$\underline{v}^+(\mathcal{R}_1)$	$\overline{v}(\mathcal{R}_1)$	**Time(s)**
v30_d10	44	40	40	21720316	0.000009	40.000	40.00	**40**	0
v30_d20	87	52	48	9864840	0.000022	48.000	48.00	**48**	0
v30_d30	131	78	55	7366010	0.000017	55.000	55.00	**55**	0
v30_d50	218	179	41	3943431	0.000049	38.667	40.00	**40**	0
v30_d70	305	444	27	3305871	0.000044	24.473	25.71	**26**	30
v50_d5	61	57	57	9434031	0.000012	57.000	57.00	**57**	0
v50_d10	123	97	96	5163842	0.000040	96.000	96.00	**96**	0
v50_d20	245	152	119	2897257	0.000065	119.000	119.00	**119**	0
v50_d30	368	280	118	1764383	0.000063	114.750	115.00	**115**	0
v50_d50	613	1033	88	1126946	0.000155	70.174	74.00	**74**	1331
v50_d70	858	5134	60	815954	0.000212	39.794	41.43	50	–
v70_d5	121	113	113	7096749	0.000026	113.000	113.00	**113**	0
v70_d10	242	174	172	2547715	0.000076	172.000	172.00	**172**	0
v70_d20	483	332	188	1227904	0.000144	185.500	186.00	**186**	0
v70_d30	725	688	197	1123376	0.000135	172.770	176.00	**176**	22
v70_d50	1208	3704	158	486048	0.000349	109.080	111.42	128	–
v70_d70	1691	38885	95	343173	0.000519	53.344	53.85	79	–
v100_d5	248	208	208	1890783	0.000107	208.000	208.00	**208**	0
v100_d10	495	351	326	1111062	0.000191	326.000	326.00	**326**	0
v100_d20	990	843	355	765677	0.000206	328.240	329.00	**329**	0
v100_d30	1485	1968	351	534777	0.000311	281.278	285.27	295	–
v100_d50	2475	17001	271	211864	0.000831	162.894	164.16	209	–
v100_d70	3465	428698	169	144826	0.001167	80.119	80.16	147	3591
v120_d5	357	319	319	1091331	0.000167	319.000	319.00	**319**	0
v120_d10	714	478	433	676061	0.000162	433.000	433.00	**433**	0
v120_d20	1428	1345	490	342343	0.000510	437.045	440.00	**440**	19
v120_d30	2142	3542	475	233983	0.000759	355.414	359.47	387	–
v120_d50	3570	38086	380	131239	0.001284	217.520	218.19	337	–
v120_d70	4998	1557394	225	94436	0.001898	99.775	∼	∼	∼
v150_d5	559	471	471	837772	0.000158	471.000	471.00	**471**	0
v150_d10	1118	732	618	418257	0.000496	617.000	617.00	**617**	0
v150_d20	2235	2468	709	188656	0.000925	598.539	603.18	609	–
v150_d30	3353	7499	690	128775	0.001467	485.285	487.25	547	–
v150_d50	5588	112736	531	71779	0.001635	282.039	282.10	492	3583
v150_d70	7823	8037920	315	48005	0.003765	129.262	∼	∼	∼
v200_d5	995	755	751	325820	0.000530	751.000	751.00	**751**	0
v200_d10	1990	1411	999	176258	0.000981	972.000	972.00	**972**	0
v200_d20	3980	5404	1143	123518	0.001773	900.160	903.49	938	–
v200_d30	5970	20434	1115	63081	0.002781	726.444	728.14	902	–
v200_d50	9950	482468	849	34897	0.005304	412.049	0.00	1811	3601
v200_d70	13930	87326813	486	21867	0.008406	∼	∼	∼	∼

Table C.1: Results for BKG and CGM-X.

				R-SCP-ECCP formulation for 3/30/300/3000 CGM calls											
				$\lvert\mathcal{C}'\rvert$				$\overline{v}(\mathcal{R}_2)$				Time(s)			
Instance	$\lvert E\rvert$	$\lvert\mathcal{C}\rvert$	$\underline{v}(\mathcal{R}_1)$	**3**	**30**	**300**	**3000**	**3**	**30**	**300**	**3000**	**3**	**30**	**300**	**3000**
v30_d10	44	40	40.000	40	40	40	40	40	40	40	40	0	0	0	0
v30_d20	87	52	48.000	50	52	52	52	48	48	48	48	0	0	0	0
v30_d30	131	78	55.000	71	76	78	78	55	55	55	55	0	0	0	0
v30_d50	218	179	38.667	102	177	178	178	43	40	40	40	0	0	0	0
v30_d70	305	444	24.473	88	334	442	441	30	26	26	26	0	12	12	12
v50_d5	61	57	57.000	57	57	57	57	57	57	57	57	0	0	0	0
v50_d10	123	97	96.000	96	97	97	97	96	96	96	96	0	0	0	0
v50_d20	245	152	119.000	140	152	152	152	119	119	119	119	0	0	0	0
v50_d30	368	280	114.750	205	278	280	280	118	115	115	115	0	0	0	0
v50_d50	613	1033	70.174	253	793	1016	1028	85	75–76	74	74	5	–	1094	978
v50_d70	858	5134	39.794	202	1289	3613	4855	60	44–52	42–50	42–50	293	–	–	–
v70_d5	121	113	113.000	113	113	113	113	113	113	113	113	0	0	0	0
v70_d10	242	174	172.000	174	174	174	174	172	172	172	172	0	0	0	0
v70_d20	483	332	185.500	259	328	330	332	189	186	186	186	0	0	0	0
v70_d30	725	688	172.770	383	667	688	686	189	176	176	176	0	21	21	59
v70_d50	1208	3704	109.080	469	2145	3511	3669	145–146	113–129	112–126	112–127	–	–	–	–
v70_d70	1691	38885	53.344	328	2721	14569	32181	92–97	59–82	55–81	54–82	–	–	–	–
v100_d5	248	208	208.000	208	208	208	208	208	208	208	208	0	0	0	0
v100_d10	495	351	326.000	337	350	351	351	326	326	326	326	0	0	0	0
v100_d20	990	843	328.240	583	828	843	843	342	329	329	329	0	0	0	0
v100_d30	1485	1968	281.278	825	1795	1961	1965	322	286–294	286–294	286–295	167	–	–	–
v100_d50	2475	17001	162.894	841	5423	13993	16853	235–253	169–215	165–210	165–208	–	–	–	–
v100_d70	3465	428698	80.119	546	5163	40665	186370	146–173	92–153	82–146	81–148	–	–	–	–
v120_d5	357	319	319.000	319	319	319	319	319	319	319	319	0	0	0	0
v120_d10	714	478	433.000	462	477	478	478	433	433	433	433	0	0	0	0
v120_d20	1428	1345	437.045	867	1325	1345	1344	459	440	440	440	0	17	18	28
v120_d30	2142	3542	355.414	1189	3129	3538	3539	418–419	360–383	360–385	360–384	–	–	–	–
v120_d50	3570	38086	217.520	1178	8710	28040	37288	317–368	227–316	219–362	219–340	–	–	–	–
v120_d70	4998	1557394	99.775	713	6896	60687	379185	193–229	119–207	104–212	100–211	–	–	–	–
v150_d5	559	471	471.000	471	471	471	471	471	471	471	471	0	0	0	0
v150_d10	1118	732	617.000	676	729	732	731	618	617	617	617	0	0	0	0
v150_d20	2235	2468	598.539	1418	2401	2466	2467	642	604–609	604–609	604–608	2	–	–	–
v150_d30	3353	7499	485.285	1797	5962	7459	7494	588–605	489–541	488–549	488–543	–	–	–	–
v150_d50	5588	112736	282.039	1647	13831	62096	105648	446–537	301–452	283–530	283–488	–	–	–	–
v150_d70	7823	8037920	129.262	999	9757	91105	717001	261–322	161–318	137–317	130–316	–	–	–	–
v200_d5	995	755	751.000	753	755	755	755	751	751	751	751	0	0	0	0
v200_d10	1990	1411	972.000	1218	1403	1411	1411	979	972	972	972	0	0	0	0
v200_d20	3980	5404	900.160	2494	5122	5402	5404	1006	904–946	903–943	903–946	–	–	–	–
v200_d30	5970	20434	726.444	3094	13522	20067	20411	923–980	734–904	729–896	729–892	–	–	–	–
v200_d50	9950	482468	412.049	2576	24034	155081	399987	695–846	457–844	415–845	413–851	–	–	–	–
v200_d70	13930	87326813	∼	1518	14939	143913	729911	399–491	241–493	204–491	191–484	–	–	–	–

Table C.2: R-SCP-ECCP formulation details and CSH results.

Instance	$\|E\|$	$\|\mathcal{C}\|$	$\underline{v}(\mathcal{R}_1)$	$\|\mathcal{C}'\|$	$\longrightarrow$	$\|\mathcal{C}'_+\|$	$\underline{v}(\mathcal{R}_2^+)$	$\overline{v}$(**CGM**)	$\longrightarrow$	$\overline{v}(\mathcal{R}_2)$	**CGA(t)**	**R-SCP-ECCP(t)**
v30_d10	44	40	40.000	40		40	40.0	40		**40**	0.00	0.00
v30_d20	87	52	48.000	50		51	48.0	50		**48**	0.00	0.00
v30_d30	131	78	55.000	57		65	55.0	57		**55**	0.00	0.00
v30_d50	218	179	38.667	47		127	38.7	47		40	0.05	0.07
v30_d70	305	444	24.473	33		147	24.5	33		27	0.13	1.06
v50_d5	61	57	57.000	57		57	57.0	57		**57**	0.00	0.00
v50_d10	123	97	96.000	96		97	96.0	96		**96**	0.00	0.00
v50_d20	245	152	119.000	122		136	119.0	122		**119**	0.00	0.00
v50_d30	368	280	114.750	131		213	114.8	131		**115**	0.02	0.01
v50_d50	613	1033	70.174	105		415	70.2	105		75	1.12	306.77
v50_d70	858	5134	39.794	70		611	39.8	70		52	8.30	–
v70_d5	121	113	113.000	113		113	113.0	113		**113**	0.00	0.00
v70_d10	242	174	172.000	174		174	172.0	174		**172**	0.00	0.00
v70_d20	483	332	185.500	202		264	185.5	202		**186**	0.02	0.00
v70_d30	725	688	172.770	209		460	172.8	209		176	0.38	11.80
v70_d50	1208	3704	109.080	173		1009	109.1	173		128	16.84	–
v70_d70	1691	38885	53.344	108		1568	53.3	108		80	210.34	–
v100_d5	248	208	208.000	208		208	208.0	208		**208**	0.00	0.00
v100_d10	495	351	326.000	327		334	326.0	327		**326**	0.00	0.00
v100_d20	990	843	328.240	382		622	328.2	382		330	0.23	0.15
v100_d30	1485	1968	281.278	370		1129	281.3	370		294	7.33	–
v100_d50	2475	17001	162.894	287		2361	162.9	287		215	412.86	–
v100_d70	3465	428698	80.119	189		1757	83.6*	189		149	–	–
v120_d5	357	319	319.000	319		319	319.0	319		**319**	0.01	0.00
v120_d10	714	478	433.000	436		449	433.0	436		**433**	0.01	0.00
v120_d20	1428	1345	437.045	513		976	437.0	513		440	0.90	7.67
v120_d30	2142	3542	355.414	502		1773	355.4	502		391	37.05	–
v120_d50	3570	38086	217.520	406		3147	218.3*	406		315	–	–
v120_d70	4998	1557394	99.775	242		889	127.7*	242		219	–	–
v150_d5	559	471	471.000	471		471	471.0	471		**471**	0.00	0.00
v150_d10	1118	732	617.000	628		669	617.0	628		**617**	0.03	0.00
v150_d20	2235	2468	598.539	751		1726	598.5	751		609	7.90	–
v150_d30	3353	7499	485.285	720		3023	485.3	720		553	283.26	–
v150_d50	5588	112736	282.039	559		2557	306.6*	559		481	–	–
v150_d70	7823	8037920	129.262	331		442	311.9*	331		324	–	20.70
v200_d5	995	755	751.000	751		753	751.0	751		**751**	0.01	0.00
v200_d10	1990	1411	972.000	1028		1169	972.0	1028		**972**	0.20	0.00
v200_d20	3980	5404	900.160	1199		3351	900.2	1199		938	108.08	–
v200_d30	5970	20434	726.444	1147		4236	741.7*	1147		893	–	–
v200_d50	9950	482468	412.049	881		2470	538.2*	881		846	–	–
v200_d70	13930	87326813	$\sim$	500		508	499.0*	500		499	–	0.01

Table C.3: RCSH results.

Instance	$\|E\|$	$\|\mathcal{C}\|$	$\underline{v}(\mathcal{R}_1)$	$\|\mathcal{C}_1\|$	$\longrightarrow$	$\|\mathcal{C}_2\|$	$\longrightarrow$	$\|\mathcal{C}_3\|$	i_1	i_2	$\underline{v}(\mathcal{R}_2)$	$\bar{v}_1(\mathcal{R}_2)$	$\longrightarrow$	$\bar{v}_2(\mathcal{R}_2)$	**Time(s)**
v30_d10	44	40	40.000	40		40		40	0	0	40.0	40		**40**(0)	0.00
v30_d20	87	52	48.000	48		48		48	0	0	48.0	48		**48**(0)	0.00
v30_d30	131	78	55.000	59		64		64	0	0	55.0	55		**55**(0)	0.00
v30_d50	218	179	38.667	50		125		128	1	9	38.7	40		40(0)	0.19
v30_d70	305	444	24.473	33		144		157	1	9	24.5	27		27(0)	2.63
v50_d5	61	57	57.000	57		57		57	0	0	57.0	57		**57**(0)	0.01
v50_d10	123	97	96.000	97		97		97	0	0	96.0	96		**96**(0)	0.00
v50_d20	245	152	119.000	123		137		137	0	0	119.0	119		**119**(0)	0.00
v50_d30	368	280	114.750	132		205		205	1	0	114.8	115		**115**(0)	0.02
v50_d50	613	1033	70.174	101		418		456	1	9	70.2	75		75(0)	309.46
v50_d70	858	5134	39.794	71		630		719	1	9	39.8	52		52(0)	1223.34
v70_d5	121	113	113.000	113		113		113	0	0	113.0	113		**113**(0)	0.00
v70_d10	242	174	172.000	174		174		174	0	0	172.0	172		**172**(0)	0.01
v70_d20	483	332	185.500	203		261		261	1	0	185.5	186		**186**(0)	0.02
v70_d30	725	688	172.770	216		469		503	1	9	172.8	176		176(0)	18.61
v70_d50	1208	3704	109.080	177		1017		1261	4	9	109.1	129		125(−4)	1394.29
v70_d70	1691	38885	53.344	110		1576		2283	4	9	53.3	83		78(−5)	4145.51
v100_d5	248	208	208.000	208		208		208	0	0	208.0	208		**208**(0)	0.00
v100_d10	495	351	326.000	328		335		335	0	0	326.0	326		**326**(0)	0.00
v100_d20	990	843	328.240	381		625		652	2	1	328.2	330		**329**(−1)	0.54
v100_d30	1485	1968	281.278	372		1122		1230	2	9	281.3	295		294(−1)	6602.07
v100_d50	2475	17001	162.894	290		2362		3205	5	9	162.9	226		211(−15)	7651.89
v100_d70	3465	428698	80.119	182		1739		3827	4	9	83.6*	163		149(−14)	9829.13
v120_d5	357	319	319.000	319		319		319	0	0	319.0	319		**319**(0)	0.01
v120_d10	714	478	433.000	441		455		455	0	0	433.0	433		**433**(0)	0.01
v120_d20	1428	1345	437.045	515		999		1050	1	9	437.0	440		440(0)	36.04
v120_d30	2142	3542	355.414	517		1777		2093	4	9	355.4	388		382(−6)	12028.63
v120_d50	3570	38086	217.520	395		3167		5614	8	9	218.3*	334		295(−39)	13336.06
v120_d70	4998	1557394	99.775	240		879		4350	6	1	129.5*	229		205(−24)	–
v150_d5	559	471	471.000	471		471		471	0	0	471.0	471		**471**(0)	0.00
v150_d10	1118	732	617.000	634		673		673	0	0	617.0	617		**617**(0)	0.03
v150_d20	2235	2468	598.539	743		1738		1859	2	9	598.5	609		608(−1)	6568.86
v150_d30	3353	7499	485.285	728		3022		3788	6	9	485.3	565		534(−31)	14675.77
v150_d50	5588	112736	282.039	558		2584		9415	11	1	306.2*	531		432(−99)	–
v150_d70	7823	8037920	129.262	329		438		1035	3	5	307.5*	319		311(−8)	–
v200_d5	995	755	751.000	751		753		753	0	0	751.0	751		**751**(0)	0.01
v200_d10	1990	1411	972.000	1023		1186		1186	0	0	972.0	972		**972**(0)	0.22
v200_d20	3980	5404	900.160	1199		3371		3762	4	9	900.2	949		933(−16)	13250.62
v200_d30	5970	20434	726.444	1163		4241		8055	9	5	741.2*	939		841(−98)	–
v200_d50	9950	482468	412.049	876		2465		5683	7	1	537.8*	855		756(−99)	–
v200_d70	13930	87326813	∼	504		513		590	4	6	502.0*	502		496(−6)	–

Table C.4: LS-RCSH results.

Appendix D

Tables of Experiments for Part II

		$K(G)$						
Instance	$\|E\|$	$\|\overline{V}_0\|$	$\|\overline{V}_{\text{iso}}\|$	$\|\overline{V}\|$	$\|\overline{E}\|$	$\overline{d}$	$\|\mathcal{C}\|$	Time(s)
v30_d10	44	44	38	6	6	0.4000	2	0.000
v30_d20	87	87	20	67	108	0.0488	32	0.000
v30_d30	131	131	6	125	327	0.0422	72	0.000
v30_d50	218	218	0	218	2,619	0.1107	179	0.002
v30_d70	305	305	0	305	13,530	0.2918	444	0.011
v50_d5	61	61	55	6	6	0.4000	2	0.000
v50_d10	123	123	81	42	60	0.0697	16	0.000
v50_d20	245	245	38	207	546	0.0256	114	0.000
v50_d30	368	368	2	366	1,920	0.0287	278	0.001
v50_d50	613	613	0	613	20,001	0.1066	1,033	0.018
v50_d70	858	858	0	858	103,218	0.2807	5,134	0.105
v70_d5	121	121	109	12	12	0.1818	4	0.000
v70_d10	242	242	124	118	162	0.0235	50	0.000
v70_d20	483	483	26	457	1,563	0.0150	306	0.001
v70_d30	725	725	3	722	6,735	0.0259	685	0.005
v70_d50	1,208	1,208	0	1,208	66,588	0.0913	3,704	0.066
v70_d70	1,691	1,691	0	1,691	389,049	0.2723	38,885	0.427
v100_d5	248	248	186	62	66	0.0349	22	0.001
v100_d10	495	495	206	289	471	0.0113	145	0.001
v100_d20	990	990	13	977	4,857	0.0102	830	0.004
v100_d30	1,485	1,485	0	1,485	22,074	0.0200	1,968	0.019
v100_d50	2,475	2,475	0	2,475	263,973	0.0862	17,001	0.303
v100_d70	3,465	3,465	0	3,465	1,581,582	0.2635	428,698	1.921
v120_d5	357	357	296	61	69	0.0377	23	0.000
v120_d10	714	714	226	488	870	0.0073	252	0.001
v120_d20	1,428	1,428	12	1,416	8,214	0.0082	1,333	0.008
v120_d30	2,142	2,142	0	2,142	41,727	0.0182	3,542	0.042
v120_d50	3,570	3,570	0	3,570	493,824	0.0775	38,086	0.494
v120_d70	4,998	4,998	0	4,998	3,235,839	0.2591	1557394	4.358
v150_d5	559	559	418	141	171	0.0173	53	0.000
v150_d10	1,118	1,118	246	872	1,623	0.0043	486	0.003
v150_d20	2,235	2,235	4	2,231	17,073	0.0069	2,464	0.014
v150_d30	3,353	3,353	0	3,353	92,262	0.0164	7,499	0.105
v150_d50	5,588	5,588	0	5,588	1,183,398	0.0758	112,736	1.672
v150_d70	7,823	7,823	0	7,823	7,783,854	0.2544	8037920	28.367
v200_d5	995	995	588	407	501	0.0061	167	0.003
v200_d10	1,990	1,990	292	1,698	4,113	0.0029	1,119	0.022
v200_d20	3,980	3,980	0	3,980	44,031	0.0056	5,404	0.053
v200_d30	5,970	5,970	0	5,970	252,393	0.0142	20,434	0.334
v200_d50	9,950	9,950	0	9,950	3,546,237	0.0716	482,468	5.393
v200_d70	13,930	13,930	0	13,930	24,489,456	0.2524	87326813	94.263

Table D.1: Properties of the transformed graph $K(G)$.

	VCCP					ECCP	
Instance	$\overline{v}(\mathcal{R}_5)$	$\underline{v}^+(\mathcal{R}_5)$	gap	Nodes	Time(s)	$\overline{v}(\mathcal{R}_1)$	$\underline{v}^+(\mathcal{R}_1)$
v30_d10	2.00	2.00	**0**	0	0.004	**40.00**	40.00
v30_d20	28.00	28.00	**0**	0	0.002	**48.00**	48.00
v30_d30	49.00	49.00	**0**	0	0.005	**55.00**	55.00
v30_d50	40.00	40.00	**0**	13	0.044	**40.00**	40.00
v30_d70	26.00	26.00	**0**	685	1.789	**26.00**	26.00
v50_d5	2.00	2.00	**0**	0	0.005	**57.00**	57.00
v50_d10	15.00	15.00	**0**	0	0.004	**96.00**	96.00
v50_d20	81.00	81.00	**0**	0	0.010	**119.00**	119.00
v50_d30	113.00	113.00	**0**	1	0.019	**115.00**	115.00
v50_d50	74.00	74.00	**0**	57,651	412.194	**74.00**	74.00
v50_d70	51.00	41.00	24.4	75,069	-	51.00	41.00
v70_d5	4.00	4.00	**0**	0	0.005	**113.00**	113.00
v70_d10	48.00	48.00	**0**	0	0.004	**172.00**	172.00
v70_d20	160.00	160.00	**0**	1	0.014	**186.00**	186.00
v70_d30	173.00	173.00	**0**	1,111	3.487	**176.00**	176.00
v70_d50	129.00	111.00	16.2	49,513	-	129.00	111.00
v70_d70	81.00	54.00	50.0	2,552	-	81.00	54.00
v100_d5	22.00	22.00	**0**	0	0.010	**208.00**	208.00
v100_d10	120.00	120.00	**0**	0	0.005	**326.00**	326.00
v100_d20	316.00	316.00	**0**	7	0.080	**329.00**	329.00
v100_d30	292.00	284.00	2.8	111,613	-	292.00	284.00
v100_d50	217.00	164.00	32.3	641	-	217.00	164.00
v100_d70	148.00	81.00	82.7	1	-	148.00	81.00
v120_d5	23.00	23.00	**0**	0	0.000	**319.00**	319.00
v120_d10	207.00	207.00	**0**	0	0.000	**433.00**	433.00
v120_d20	428.00	428.00	**0**	5,002	18.400	**440.00**	440.00
v120_d30	385.00	358.00	7.5	30,883	-	385.00	358.00
v120_d50	358.00	218.00	64.2	48	-	358.00	218.00
v120_d70	~	~	~	~	~	~	~
v150_d5	53.00	53.00	**0**	0	0.012	**471.00**	471.00
v150_d10	371.00	371.00	**0**	0	0.020	**617.00**	617.00
v150_d20	605.00	598.00	1.2	123,732	-	609.00	602.00
v150_d30	550.00	486.00	13.2	4,785	-	550.00	486.00
v150_d50	1133.00	283.00	300.4	1	-	1133.00	283.00
v150_d70	~	~	~	~	~	~	~
v200_d5	163.00	163.00	**0**	0	0.014	**751.00**	751.00
v200_d10	680.00	680.00	**0**	0	0.074	**972.00**	972.00
v200_d20	938.00	901.00	4.1	12,406	-	938.00	901.00
v200_d30	922.00	727.00	26.8	47	-	922.00	727.00
v200_d50	2010.00	0.00	-	1	-	2010.00	0.00
v200_d70	~	~	~	~	~	~	~

Table D.2: Experiment 1. VCCP and ECCP Solution Quality

Instance	Rows	Columns	Non-zeros
v30_d10	12	12	24
v30_d20	212	175	503
v30_d30	718	452	2,103
v30_d50	10,231	2,837	49,416
v30_d70	145,026	13,835	883,532
v50_d5	12	12	24
v50_d10	118	102	280
v50_d20	1,080	753	2,994
v50_d30	4,969	2,286	18,374
v50_d50	184,749	20,614	1,512,471
v50_d70	3,503,178	104,076	32,129,604
v70_d5	24	24	48
v70_d10	315	280	712
v70_d20	3,257	2,020	9,607
v70_d30	23,992	7,457	115,277
v70_d50	1,079,613	67,796	12,009,301
v70_d70	30,118,550	390,740	349,487,629
v100_d5	134	128	278
v100_d10	917	760	2,216
v100_d20	12,797	5,834	49,056
v100_d30	104,605	23,559	688,700
v100_d50	9,183,251	266,448	142,696,577
v100_d70	∼	∼	∼
v120_d5	136	130	286
v120_d10	1,789	1,358	4,655
v120_d20	22,087	9,630	87,070
v120_d30	246,806	43,869	2,024,933
v120_d50	23,355,191	497,394	463,057,317
v120_d70	∼	∼	∼
v150_d5	342	312	744
v150_d10	3,231	2,495	8,312
v150_d20	53,470	19,304	248,609
v150_d30	757,473	95,615	8,402,812
v150_d50	∼	∼	∼
v150_d70	∼	∼	∼
v200_d5	1,012	908	2,239
v200_d10	8,717	5,811	25,330
v200_d20	172,937	48,011	1,037,715
v200_d30	2,939,611	258,363	45,472,126
v200_d50	∼	∼	∼
v200_d70	∼	∼	∼

Table D.3: Model Dimension Table for VCCP formulation based on representatives.

	VCCP					ECCP	
Instance	$\underline{v}(\mathcal{R}_{10})$	$\overline{v}(\mathcal{R}_{10})$	$\underline{v}(\mathcal{R}_{10}^{+})$	Gap	Time (s)	$\overline{v}(\mathcal{R}_1)$	$\underline{v}^{+}(\mathcal{R}_1)$
v30_d10	2.00	2.00	2.00	**0**	0.1	**40.00**	40.00
v30_d20	28.00	28.00	28.00	**0**	0.0	**48.00**	48.00
v30_d30	48.50	49.00	49.00	**0**	0.0	**55.00**	55.00
v30_d50	36.87	40.00	40.00	**0**	61.3	**40.00**	40.00
v30_d70	21.92	28.00	23.00	21.74	-	28.00	23.00
v50_d5	2.00	2.00	2.00	**0**	0.0	**57.00**	57.00
v50_d10	15.00	15.00	15.00	**0**	0.0	**96.00**	96.00
v50_d20	81.00	81.00	81.00	**0**	0.0	**119.00**	119.00
v50_d30	111.00	113.00	113.00	**0**	0.7	**115.00**	115.00
v50_d50	63.70	82.00	66.00	24.24	-	82.00	66.00
v50_d70	~	~	~	~	~	~	~
v70_d5	4.00	4.00	4.00	**0**	0.0	**113.00**	113.00
v70_d10	48.00	48.00	48.00	**0**	0.0	**172.00**	172.00
v70_d20	159.25	160.00	160.00	**0**	0.1	**186.00**	186.00
v70_d30	164.44	173.00	171.00	1.17	1854.6	176.00	174.00
v70_d50	~	~	~	~	~	~	~
v70_d70	~	~	~	~	~	~	~
v100_d5	22.00	22.00	22.00	**0**	0.0	**208.00**	208.00
v100_d10	120.00	120.00	120.00	**0**	0.0	**326.00**	326.00
v100_d20	308.71	316.00	316.00	**0**	9.9	**329.00**	329.00
v100_d30	263.13	303.00	269.00	12.64	-	303.00	269.00
v100_d50	~	~	~	~	~	~	~
v100_d70	~	~	~	~	~	~	~
v120_d5	23.00	23.00	23.00	**0**	0.0	**319.00**	319.00
v120_d10	207.00	207.00	207.00	**0**	0.0	**433.00**	433.00
v120_d20	416.37	428.00	427.00	0.23	1845.5	440.00	439.00
v120_d30	~	~	~	~	~	~	~
v120_d50	~	~	~	~	~	~	~
v120_d70	~	~	~	~	~	~	~
v150_d5	53.00	53.00	53.00	**0**	0.0	**471.00**	471.00
v150_d10	370.50	371.00	371.00	**0**	0.1	**617.00**	617.00
v150_d20	574.81	608.00	595.00	2.18	3548.5	612.00	599.00
v150_d30	~	~	~	~	~	~	~
v150_d50	~	~	~	~	~	~	~
v150_d70	~	~	~	~	~	~	~
v200_d5	163.00	163.00	163.00	**0**	0.0	**751.00**	751.00
v200_d10	679.50	680.00	680.00	**0**	0.3	**972.00**	972.00
v200_d20	~	~	~	~	~	~	~
v200_d30	~	~	~	~	~	~	~
v200_d50	~	~	~	~	~	~	~
v200_d70	~	~	~	~	~	~	~

Table D.4: Experiment 2. Solution Quality

Instance	Nodes	Root Cuts	Node Cuts
v30_d10	0	0	0
v30_d20	0	0	0
v30_d30	1	4	0
v30_d50	981	277	0
v30_d70	30	87	0
v50_d5	0	0	0
v50_d10	0	0	0
v50_d20	1	0	0
v50_d30	1	42	0
v50_d50	14	50	0
v50_d70	~	~	~
v70_d5	0	0	0
v70_d10	0	0	0
v70_d20	1	2	0
v70_d30	10,777	336	0
v70_d50	~	~	~
v70_d70	~	~	~
v100_d5	0	0	0
v100_d10	0	0	0
v100_d20	272	115	0
v100_d30	63	90	0
v100_d50	~	~	~
v100_d70	~	~	~
v120_d5	0	0	0
v120_d10	1	0	0
v120_d20	18,632	259	0
v120_d30	~	~	~
v120_d50	~	~	~
v120_d70	~	~	~
v150_d5	0	0	0
v150_d10	0	1	0
v150_d20	674	966	0
v150_d30	~	~	~
v150_d50	~	~	~
v150_d70	~	~	~
v200_d5	0	0	0
v200_d10	1	2	0
v200_d20	~	~	~
v200_d30	~	~	~
v200_d50	~	~	~
v200_d70	~	~	~

Table D.5: Experiment 2. Branch-and-Cut Statistics

	VCCP					ECCP	
Instance	$\underline{v}(\mathcal{R}_{10})$	$\overline{v}(\mathcal{R}_{10})$	$\underline{v}(\mathcal{R}_{10}^{+})$	Gap	Time (s)	$\overline{v}(\mathcal{R}_1)$	$\underline{v}^{+}(\mathcal{R}_1)$
v30_d10	2.00	2.00	2.00	**0**	0.0	**40.00**	40.00
v30_d20	28.00	28.00	28.00	**0**	0.0	**48.00**	48.00
v30_d30	48.50	49.00	49.00	**0**	0.0	**55.00**	55.00
v30_d50	36.87	40.00	40.00	**0**	165.1	**40.00**	40.00
v30_d70	21.92	27.00	24.00	12.5	-	27.00	24.00
v50_d5	2.00	2.00	2.00	**0**	0.0	**57.00**	57.00
v50_d10	15.00	15.00	15.00	**0**	0.0	**96.00**	96.00
v50_d20	81.00	81.00	81.00	**0**	0.0	**119.00**	119.00
v50_d30	111.00	113.00	113.00	**0**	0.6	**115.00**	115.00
v50_d50	63.70	84.00	66.00	27.27	-	84.00	66.00
v50_d70	~	~	~	~	~	~	~
v70_d5	4.00	4.00	4.00	**0**	0.0	**113.00**	113.00
v70_d10	48.00	48.00	48.00	**0**	0.0	**172.00**	172.00
v70_d20	159.25	160.00	160.00	**0**	0.1	**186.00**	186.00
v70_d30	164.44	173.00	171.00	1.17	1835.3	176.00	174.00
v70_d50	~	~	~	~	~	~	~
v70_d70	~	~	~	~	~	~	~
v100_d5	22.00	22.00	22.00	**0**	0.0	**208.00**	208.00
v100_d10	120.00	120.00	120.00	**0**	0.0	**326.00**	326.00
v100_d20	308.71	316.00	316.00	**0**	9.9	**329.00**	329.00
v100_d30	263.13	305.00	275.00	10.91	-	305.00	275.00
v100_d50	~	~	~	~	~	~	~
v100_d70	~	~	~	~	~	~	~
v120_d5	23.00	23.00	23.00	**0**	0.0	**319.00**	319.00
v120_d10	207.00	207.00	207.00	**0**	0.0	**433.00**	433.00
v120_d20	416.37	428.00	427.00	0.23	1837.2	440.00	439.00
v120_d30	~	~	~	~	~	~	~
v120_d50	~	~	~	~	~	~	~
v120_d70	~	~	~	~	~	~	~
v150_d5	53.00	53.00	53.00	**0**	0.0	**471.00**	471.00
v150_d10	370.50	371.00	371.00	**0**	0.0	**617.00**	617.00
v150_d20	574.81	608.00	595.00	2.19	2740.8	612.00	599.00
v150_d30	~	~	~	~	~	~	~
v150_d50	~	~	~	~	~	~	~
v150_d70	~	~	~	~	~	~	~
v200_d5	163.00	163.00	163.00	**0**	0.0	**751.00**	751.00
v200_d10	679.50	680.00	680.00	**0**	0.2	**972.00**	972.00
v200_d20	~	~	~	~	~	~	~
v200_d30	~	~	~	~	~	~	~
v200_d50	~	~	~	~	~	~	~
v200_d70	~	~	~	~	~	~	~

Table D.6: Experiment 3. Solution Quality

Instance	Nodes	Root Cuts	Node Cuts
v30_d10	0	0	0
v30_d20	0	0	0
v30_d30	1	5	0
v30_d50	4,629	406	0
v30_d70	11	427	0
v50_d5	0	0	0
v50_d10	0	0	0
v50_d20	0	0	0
v50_d30	1	44	0
v50_d50	11	153	0
v50_d70	∼	∼	∼
v70_d5	0	0	0
v70_d10	0	0	0
v70_d20	1	2	0
v70_d30	9,198	393	0
v70_d50	∼	∼	∼
v70_d70	∼	∼	∼
v100_d5	0	0	0
v100_d10	0	0	0
v100_d20	289	207	0
v100_d30	11	581	0
v100_d50	∼	∼	∼
v100_d70	∼	∼	∼
v120_d5	0	0	0
v120_d10	0	0	0
v120_d20	16,224	312	0
v120_d30	∼	∼	∼
v120_d50	∼	∼	∼
v120_d70	∼	∼	∼
v150_d5	0	0	0
v150_d10	0	1	0
v150_d20	521	1,191	0
v150_d30	∼	∼	∼
v150_d50	∼	∼	∼
v150_d70	∼	∼	∼
v200_d5	0	0	0
v200_d10	0	2	0
v200_d20	∼	∼	∼
v200_d30	∼	∼	∼
v200_d50	∼	∼	∼
v200_d70	∼	∼	∼

Table D.7: Experiment 3. Branch-and-Cut Statistics

	VCCP					ECCP	
Instance	$\underline{v}(\mathcal{R}_{10})$	$\overline{v}(\mathcal{R}_{10})$	$\underline{v}(\mathcal{R}_{10}^{+})$	Gap	Time (s)	$\overline{v}(\mathcal{R}_1)$	$\underline{v}^{+}(\mathcal{R}_1)$
v30_d10	2.00	2.00	2.00	**0**	0.0	**40.00**	40.00
v30_d20	28.00	28.00	28.00	**0**	0.0	**48.00**	48.00
v30_d30	48.50	49.00	49.00	**0**	0.0	**55.00**	55.00
v30_d50	36.87	40.00	40.00	**0**	75.3	**40.00**	40.00
v30_d70	21.92	28.00	24.00	16.67	-	28.00	24.00
v50_d5	2.00	2.00	2.00	**0**	0.0	**57.00**	57.00
v50_d10	15.00	15.00	15.00	**0**	0.0	**96.00**	96.00
v50_d20	81.00	81.00	81.00	**0**	0.0	**119.00**	119.00
v50_d30	111.00	113.00	113.00	**0**	0.5	**115.00**	115.00
v50_d50	~	~	~	~	~	~	~
v50_d70	~	~	~	~	~	~	~
v70_d5	4.00	4.00	4.00	**0**	0.0	**113.00**	113.00
v70_d10	48.00	48.00	48.00	**0**	0.0	**172.00**	172.00
v70_d20	159.25	160.00	160.00	**0**	0.1	**186.00**	186.00
v70_d30	164.44	173.00	172.00	0.58	1832.2	176.00	175.00
v70_d50	~	~	~	~	~	~	~
v70_d70	~	~	~	~	~	~	~
v100_d5	22.00	22.00	22.00	**0**	0.0	**208.00**	208.00
v100_d10	120.00	120.00	120.00	**0**	0.0	**326.00**	326.00
v100_d20	308.71	316.00	316.00	**0**	4.4	**329.00**	329.00
v100_d30	263.13	300.00	279.00	7.53	-	300.00	279.00
v100_d50	~	~	~	~	~	~	~
v100_d70	~	~	~	~	~	~	~
v120_d5	23.00	23.00	23.00	**0**	0.0	**319.00**	319.00
v120_d10	207.00	207.00	207.00	**0**	0.0	**433.00**	433.00
v120_d20	416.37	428.00	427.00	0.23	1827.9	440.00	439.00
v120_d30	~	~	~	~	~	~	~
v120_d50	~	~	~	~	~	~	~
v120_d70	~	~	~	~	~	~	~
v150_d5	53.00	53.00	53.00	**0**	0.0	**471.00**	471.00
v150_d10	370.50	371.00	371.00	**0**	0.1	**617.00**	617.00
v150_d20	574.81	610.00	595.00	2.52	2332.7	614.00	599.00
v150_d30	~	~	~	~	~	~	~
v150_d50	~	~	~	~	~	~	~
v150_d70	~	~	~	~	~	~	~
v200_d5	163.00	163.00	163.00	**0**	0.0	**751.00**	751.00
v200_d10	679.50	680.00	680.00	**0**	0.2	**972.00**	972.00
v200_d20	~	~	~	~	~	~	~
v200_d30	~	~	~	~	~	~	~
v200_d50	~	~	~	~	~	~	~
v200_d70	~	~	~	~	~	~	~

Table D.8: Experiment 4. Solution Quality

Instance	Nodes	Root Cuts	Node Cuts
v30_d10	0	0	0
v30_d20	0	0	0
v30_d30	1	1,036	0
v30_d50	1,143	17,740	0
v30_d70	11	143,258	0
v50_d5	0	0	0
v50_d10	0	0	0
v50_d20	0	0	0
v50_d30	1	5,283	0
v50_d50	~	~	~
v50_d70	~	~	~
v70_d5	0	0	0
v70_d10	0	0	0
v70_d20	1	4,328	0
v70_d30	10,287	41,258	0
v70_d50	~	~	~
v70_d70	~	~	~
v100_d5	0	0	0
v100_d10	0	0	0
v100_d20	54	23,223	0
v100_d30	12	69,055	0
v100_d50	~	~	~
v100_d70	~	~	~
v120_d5	0	0	0
v120_d10	0	0	0
v120_d20	18,894	40,975	0
v120_d30	~	~	~
v120_d50	~	~	~
v120_d70	~	~	~
v150_d5	0	0	0
v150_d10	0	606	0
v150_d20	588	128,292	0
v150_d30	~	~	~
v150_d50	~	~	~
v150_d70	~	~	~
v200_d5	0	0	0
v200_d10	0	2,279	0
v200_d20	~	~	~
v200_d30	~	~	~
v200_d50	~	~	~
v200_d70	~	~	~

Table D.9: Experiment 4. Branch-and-Cut Statistics

	VCCP					ECCP	
Instance	$\underline{v}(\mathcal{R}_{10})$	$\overline{v}(\mathcal{R}_{10})$	$\underline{v}(\mathcal{R}^+_{10})$	Gap	Time (s)	$\overline{v}(\mathcal{R}_1)$	$\underline{v}^+(\mathcal{R}_1)$
v30_d10	2.00	2.00	2.00	**0**	0.0	**40.00**	40.00
v30_d20	28.00	28.00	28.00	**0**	0.0	**48.00**	48.00
v30_d30	48.50	49.00	49.00	**0**	0.0	**55.00**	55.00
v30_d50	36.87	40.00	40.00	**0**	95.9	**40.00**	40.00
v30_d70	21.92	29.00	23.00	26.09	-	29.00	23.00
v50_d5	2.00	2.00	2.00	**0**	0.0	**57.00**	57.00
v50_d10	15.00	15.00	15.00	**0**	0.0	**96.00**	96.00
v50_d20	81.00	81.00	81.00	**0**	0.0	**119.00**	119.00
v50_d30	111.00	113.00	113.00	**0**	0.8	**115.00**	115.00
v50_d50	63.70	82.00	66.00	24.24	-	82.00	66.00
v50_d70	~	~	~	~	~	~	~
v70_d5	4.00	4.00	4.00	**0**	0.0	**113.00**	113.00
v70_d10	48.00	48.00	48.00	**0**	0.0	**172.00**	172.00
v70_d20	159.25	160.00	160.00	**0**	0.1	**186.00**	186.00
v70_d30	164.44	173.00	172.00	0.58	1848.4	176.00	175.00
v70_d50	~	~	~	~	~	~	~
v70_d70	~	~	~	~	~	~	~
v100_d5	22.00	22.00	22.00	**0**	0.0	**208.00**	208.00
v100_d10	120.00	120.00	120.00	**0**	0.0	**326.00**	326.00
v100_d20	308.71	316.00	316.00	**0**	7.4	**329.00**	329.00
v100_d30	263.13	307.00	269.00	14.13	-	307.00	269.00
v100_d50	~	~	~	~	~	~	~
v100_d70	~	~	~	~	~	~	~
v120_d5	23.00	23.00	23.00	**0**	0.0	**319.00**	319.00
v120_d10	207.00	207.00	207.00	**0**	0.0	**433.00**	433.00
v120_d20	416.37	428.00	427.00	0.23	1851.2	440.00	439.00
v120_d30	~	~	~	~	~	~	~
v120_d50	~	~	~	~	~	~	~
v120_d70	~	~	~	~	~	~	~
v150_d5	53.00	53.00	53.00	**0**	0.0	**471.00**	471.00
v150_d10	370.50	371.00	371.00	**0**	0.1	**617.00**	617.00
v150_d20	574.81	608.00	595.00	2.19	3480.1	612.00	599.00
v150_d30	~	~	~	~	~	~	~
v150_d50	~	~	~	~	~	~	~
v150_d70	~	~	~	~	~	~	~
v200_d5	163.00	163.00	163.00	**0**	0.0	**751.00**	751.00
v200_d10	679.50	680.00	680.00	**0**	0.2	**972.00**	972.00
v200_d20	842.89	3073.00	99.00	3003.03	-	3073.00	99.00
v200_d30	~	~	~	~	~	~	~
v200_d50	~	~	~	~	~	~	~
v200_d70	~	~	~	~	~	~	~

Table D.10: Experiment 5. Solution Quality

Instance	Nodes	Root Cuts	Node Cuts
v30_d10	0	0	0
v30_d20	0	0	0
v30_d30	0	4	0
v30_d50	1,385	277	373
v30_d70	11	74	13
v50_d5	0	0	0
v50_d10	0	0	0
v50_d20	0	0	0
v50_d30	1	42	2
v50_d50	11	48	13
v50_d70	∼	∼	∼
v70_d5	0	0	0
v70_d10	0	0	0
v70_d20	1	2	0
v70_d30	9,408	336	3,582
v70_d50	∼	∼	∼
v70_d70	∼	∼	∼
v100_d5	0	0	0
v100_d10	0	0	0
v100_d20	69	115	43
v100_d30	30	86	32
v100_d50	∼	∼	∼
v100_d70	∼	∼	∼
v120_d5	0	0	0
v120_d10	0	0	0
v120_d20	17,756	259	2,514
v120_d30	∼	∼	∼
v120_d50	∼	∼	∼
v120_d70	∼	∼	∼
v150_d5	0	0	0
v150_d10	0	1	0
v150_d20	599	966	592
v150_d30	∼	∼	∼
v150_d50	∼	∼	∼
v150_d70	∼	∼	∼
v200_d5	0	0	0
v200_d10	0	2	0
v200_d20	1	1	0
v200_d30	∼	∼	∼
v200_d50	∼	∼	∼
v200_d70	∼	∼	∼

Table D.11: Experiment 5. Branch-and-Cut Statistics

	VCCP					ECCP	
Instance	$\underline{v}(\mathcal{R}_{10})$	$\overline{v}(\mathcal{R}_{10})$	$\underline{v}(\mathcal{R}^+_{10})$	Gap	Time (s)	$\overline{v}(\mathcal{R}_1)$	$\underline{v}^+(\mathcal{R}_1)$
v30_d10	2.00	2.00	2.00	**0**	0.0	**40.00**	40.00
v30_d20	28.00	28.00	28.00	**0**	0.0	**48.00**	48.00
v30_d30	48.50	49.00	49.00	**0**	0.0	**55.00**	55.00
v30_d50	36.87	40.00	40.00	**0**	120.4	**40.00**	40.00
v30_d70	21.92	28.00	24.00	16.67	-	28.00	24.00
v50_d5	2.00	2.00	2.00	**0**	0.0	**57.00**	57.00
v50_d10	15.00	15.00	15.00	**0**	0.0	**96.00**	96.00
v50_d20	81.00	81.00	81.00	**0**	0.0	**119.00**	119.00
v50_d30	111.00	113.00	113.00	**0**	0.5	**115.00**	115.00
v50_d50	63.70	82.00	67.00	22.39	-	82.00	67.00
v50_d70	~	~	~	~	~	~	~
v70_d5	4.00	4.00	4.00	**0**	0.0	**113.00**	113.00
v70_d10	48.00	48.00	48.00	**0**	0.0	**172.00**	172.00
v70_d20	159.25	160.00	160.00	**0**	0.1	**186.00**	186.00
v70_d30	164.44	173.00	172.00	0.58	1831.5	176.00	175.00
v70_d50	~	~	~	~	~	~	~
v70_d70	~	~	~	~	~	~	~
v100_d5	22.00	22.00	22.00	**0**	0.0	**208.00**	208.00
v100_d10	120.00	120.00	120.00	**0**	0.0	**326.00**	326.00
v100_d20	308.71	316.00	316.00	**0**	4.3	**329.00**	329.00
v100_d30	263.13	299.00	278.00	7.55	-	299.00	278.00
v100_d50	~	~	~	~	~	~	~
v100_d70	~	~	~	~	~	~	~
v120_d5	23.00	23.00	23.00	**0**	0.0	**319.00**	319.00
v120_d10	207.00	207.00	207.00	**0**	0.0	**433.00**	433.00
v120_d20	416.37	428.00	428.00	**0**	1294.2	**440.00**	440.00
v120_d30	~	~	~	~	~	~	~
v120_d50	~	~	~	~	~	~	~
v120_d70	~	~	~	~	~	~	~
v150_d5	53.00	53.00	53.00	**0**	0.0	**471.00**	471.00
v150_d10	370.50	371.00	371.00	**0**	0.1	**617.00**	617.00
v150_d20	574.81	608.00	595.00	2.19	2176.6	612.00	599.00
v150_d30	~	~	~	~	~	~	~
v150_d50	~	~	~	~	~	~	~
v150_d70	~	~	~	~	~	~	~
v200_d5	163.00	163.00	163.00	**0**	0.0	**751.00**	751.00
v200_d10	679.50	680.00	680.00	**0**	0.2	**972.00**	972.00
v200_d20	~	~	~	~	~	~	~
v200_d30	~	~	~	~	~	~	~
v200_d50	~	~	~	~	~	~	~
v200_d70	~	~	~	~	~	~	~

Table D.12: Experiment 6. Solution Quality

Instance	Nodes	Root Cuts	Node Cuts
v30_d10	0	0	0
v30_d20	0	0	0
v30_d30	1	5	0
v30_d50	2,213	406	373
v30_d70	18	644	20
v50_d5	0	0	0
v50_d10	0	0	0
v50_d20	0	0	0
v50_d30	1	44	0
v50_d50	11	153	13
v50_d70	∼	∼	∼
v70_d5	0	0	0
v70_d10	0	0	0
v70_d20	1	2	0
v70_d30	11,093	393	4,503
v70_d50	∼	∼	∼
v70_d70	∼	∼	∼
v100_d5	0	0	0
v100_d10	0	0	0
v100_d20	25	207	12
v100_d30	39	1,033	41
v100_d50	∼	∼	∼
v100_d70	∼	∼	∼
v120_d5	0	0	0
v120_d10	0	0	0
v120_d20	12,651	312	1,358
v120_d30	∼	∼	∼
v120_d50	∼	∼	∼
v120_d70	∼	∼	∼
v150_d5	0	0	0
v150_d10	0	1	0
v150_d20	743	1,191	695
v150_d30	∼	∼	∼
v150_d50	∼	∼	∼
v150_d70	∼	∼	∼
v200_d5	0	0	0
v200_d10	0	2	0
v200_d20	∼	∼	∼
v200_d30	∼	∼	∼
v200_d50	∼	∼	∼
v200_d70	∼	∼	∼

Table D.13: Experiment 6. Branch-and-Cut Statistics

	VCCP					ECCP	
Instance	$\underline{v}(\mathcal{R}_{10})$	$\overline{v}(\mathcal{R}_{10})$	$\underline{v}(\mathcal{R}^+_{10})$	Gap	Time (s)	$\overline{v}(\mathcal{R}_1)$	$\underline{v}^+(\mathcal{R}_1)$
v30_d10	2.00	2.00	2.00	**0**	0.0	**40.00**	40.00
v30_d20	28.00	28.00	28.00	**0**	0.0	**48.00**	48.00
v30_d30	48.50	49.00	49.00	**0**	0.0	**55.00**	55.00
v30_d50	36.87	40.00	40.00	**0**	122.3	**40.00**	40.00
v30_d70	21.92	27.00	24.00	12.50	-	27.00	24.00
v50_d5	2.00	2.00	2.00	**0**	0.0	**57.00**	57.00
v50_d10	15.00	15.00	15.00	**0**	0.0	**96.00**	96.00
v50_d20	81.00	81.00	81.00	**0**	0.0	**119.00**	119.00
v50_d30	111.00	113.00	113.00	**0**	0.5	**115.00**	115.00
v50_d50	~	~	~	~	~	~	~
v50_d70	~	~	~	~	~	~	~
v70_d5	4.00	4.00	4.00	**0**	0.0	**113.00**	113.00
v70_d10	48.00	48.00	48.00	**0**	0.0	**172.00**	172.00
v70_d20	159.25	160.00	160.00	**0**	0.1	**186.00**	186.00
v70_d30	164.44	173.00	172.00	0.58	1827.7	176.00	175.00
v70_d50	~	~	~	~	~	~	~
v70_d70	~	~	~	~	~	~	~
v100_d5	22.00	22.00	22.00	**0**	0.0	**208.00**	208.00
v100_d10	120.00	120.00	120.00	**0**	0.0	**326.00**	326.00
v100_d20	308.71	316.00	316.00	**0**	5.6	**329.00**	329.00
v100_d30	263.13	303.00	279.00	8.60	-	303.00	279.00
v100_d50	~	~	~	~	~	~	~
v100_d70	~	~	~	~	~	~	~
v120_d5	23.00	23.00	23.00	**0**	0.0	**319.00**	319.00
v120_d10	207.00	207.00	207.00	**0**	0.0	**433.00**	433.00
v120_d20	416.37	428.00	427.00	0.23	1833.3	440.00	439.00
v120_d30	~	~	~	~	~	~	~
v120_d50	~	~	~	~	~	~	~
v120_d70	~	~	~	~	~	~	~
v150_d5	53.00	53.00	53.00	**0**	0.0	**471.00**	471.00
v150_d10	370.50	371.00	371.00	**0**	0.0	**617.00**	617.00
v150_d20	574.81	608.00	595.00	2.19	2363.5	612.00	599.00
v150_d30	~	~	~	~	~	~	~
v150_d50	~	~	~	~	~	~	~
v150_d70	~	~	~	~	~	~	~
v200_d5	163.00	163.00	163.00	**0**	0.0	**751.00**	751.00
v200_d10	679.50	680.00	680.00	**0**	0.2	**972.00**	972.00
v200_d20	~	~	~	~	~	~	~
v200_d30	~	~	~	~	~	~	~
v200_d50	~	~	~	~	~	~	~
v200_d70	~	~	~	~	~	~	~

Table D.14: Experiment 7. Solution Quality

Instance	Nodes	Root Cuts	Node Cuts
v30_d10	0	0	0
v30_d20	0	0	0
v30_d30	1	1,036	0
v30_d50	1,695	17,740	157
v30_d70	10	143,258	12
v50_d5	0	0	0
v50_d10	0	0	0
v50_d20	0	0	0
v50_d30	1	5,283	0
v50_d50	∼	∼	∼
v50_d70	∼	∼	∼
v70_d5	0	0	0
v70_d10	0	0	0
v70_d20	1	4,328	0
v70_d30	10,343	41,258	3,225
v70_d50	∼	∼	∼
v70_d70	∼	∼	∼
v100_d5	0	0	0
v100_d10	0	0	0
v100_d20	54	23,223	15
v100_d30	19	69,055	21
v100_d50	∼	∼	∼
v100_d70	∼	∼	∼
v120_d5	0	0	0
v120_d10	0	0	0
v120_d20	18,524	40,975	1,160
v120_d30	∼	∼	∼
v120_d50	∼	∼	∼
v120_d70	∼	∼	∼
v150_d5	0	0	0
v150_d10	0	606	0
v150_d20	561	128,292	389
v150_d30	∼	∼	∼
v150_d50	∼	∼	∼
v150_d70	∼	∼	∼
v200_d5	0	0	0
v200_d10	0	2,279	0
v200_d20	∼	∼	∼
v200_d30	∼	∼	∼
v200_d50	∼	∼	∼
v200_d70	∼	∼	∼

Table D.15: Experiment 7. Branch-and-Cut Statistics

Instance	$\overline{v}$(CGM)	$\underline{v}(\mathcal{R}_6)$	$\underline{v}(\mathcal{R}_6^+)$	Root Time	Generated Columns	Col Gen Iterations	$\overline{\underline{v}}(\mathcal{R}_5)$
v30_d10	2.00	2.00	2.00	0.00	0	1	**2.00**
v30_d20	29.00	28.00	28.00	0.00	1	2	**28.00**
v30_d30	50.00	50.00	49.00	0.00	8	3	**49.00**
v30_d50	47.00	45.00	38.67	0.04	93	10	-
v30_d70	32.00	32.00	24.47	0.07	199	7	-
v50_d5	2.00	2.00	2.00	0.00	0	1	**2.00**
v50_d10	15.00	15.00	15.00	0.00	1	2	**15.00**
v50_d20	84.00	82.00	81.00	0.00	10	3	**81.00**
v50_d30	128.00	122.00	112.75	0.02	83	7	-
v50_d50	99.00	96.00	70.17	0.26	484	7	-
v50_d70	67.00	67.00	39.79	5.75	1,114	13	-
v70_d5	4.00	4.00	4.00	0.00	0	1	**4.00**
v70_d10	49.00	48.00	48.00	0.00	0	1	**48.00**
v70_d20	167.00	167.00	159.50	0.01	59	5	-
v70_d30	199.00	195.00	169.77	0.10	314	8	-
v70_d50	167.00	162.00	109.08	3.57	1,423	10	-
v70_d70	101.00	100.00	53.34	196.95	3,200	15	-
v100_d5	22.00	22.00	22.00	0.00	0	1	**22.00**
v100_d10	123.00	121.00	120.00	0.01	4	2	**120.00**
v100_d20	350.00	345.00	315.24	0.07	294	10	-
v100_d30	361.00	352.00	281.28	0.98	1,017	8	-
v100_d50	270.00	267.00	162.89	92.59	3,927	14	-
v100_d70	173.00	171.00	81.02*	2173.49	7,949	9	-
v120_d5	23.00	23.00	23.00	0.00	0	1	**23.00**
v120_d10	216.00	210.00	207.00	0.01	10	3	**207.00**
v120_d20	488.00	474.00	425.04	0.21	532	11	-
v120_d30	464.00	462.00	355.41	4.94	1,863	10	-
v120_d50	371.00	369.00	217.52	798.29	6,909	15	-
v120_d70	∼	∼	∼	∼	∼	∼	∼
v150_d5	53.00	53.00	53.00	0.00	0	1	**53.00**
v150_d10	377.00	372.00	371.00	0.01	36	4	**371.00**
v150_d20	710.00	691.00	594.54	1.12	1,205	9	-
v150_d30	679.00	665.00	485.29	37.72	3,602	10	-
v150_d50	517.00	513.00	284.31*	2545.64	11,945	7	-
v150_d70	∼	∼	∼	∼	∼	∼	∼
v200_d5	164.00	163.00	163.00	0.00	0	1	**163.00**
v200_d10	721.00	699.00	680.00	0.06	170	8	**680.00**
v200_d20	1115.00	1092.00	900.16	12.98	2,856	9	-
v200_d30	1067.00	1059.00	726.44	1098.20	8,324	13	-
v200_d50	802.00	800.00	468.85*	-	17,252	4	-
v200_d70	∼	∼	∼	∼	∼	∼	∼

Table D.16: Branch-and-Price. CGA Results at Root Node.

Instance	$\overline{v}(\mathcal{R}_6)$	$\underline{v}^+(\mathcal{R}_6^+)$	Gap	Nodes	Time(s)
v30_d10	**2.00**	2.00	0	0	0.028
v30_d20	**28.00**	28.00	0	0	0.004
v30_d30	**49.00**	49.00	0	0	0.009
v30_d50	**40.00**	39.19	0	18	0.409
v30_d70	27.00	25.50	5.88	1,610	-
v50_d5	**2.00**	2.00	0	0	0.006
v50_d10	**15.00**	15.00	0	0	0.006
v50_d20	**81.00**	81.00	0	0	0.006
v50_d30	**113.00**	112.75	0	6	0.080
v50_d50	77.00	71.05	8.37	1,836	-
v50_d70	54.00	40.45	33.50	329	-
v70_d5	**4.00**	4.00	0	0	0.007
v70_d10	**48.00**	48.00	0	0	0.008
v70_d20	**160.00**	159.50	0	2	0.027
v70_d30	175.00	171.02	2.33	4,071	-
v70_d50	136.00	109.93	23.70	648	-
v70_d70	82.00	53.65	52.83	22	-
v100_d5	**22.00**	22.00	0	0	0.004
v100_d10	**120.00**	120.00	0	0	0.012
v100_d20	**316.00**	315.63	0	98	3.867
v100_d30	303.00	282.42	7.29	2,015	-
v100_d50	226.00	163.29	38.39	68	-
v100_d70	171.00	81.02*	111.08	1	-
v120_d5	**23.00**	23.00	0	0	0.034
v120_d10	**207.00**	207.00	0	0	0.016
v120_d20	429.00	426.11	0.68	3,322	-
v120_d30	407.00	356.93	14.03	978	-
v120_d50	318.00	217.75	46.03	9	-
v120_d70	∼	∼	∼	∼	∼
v150_d5	**53.00**	53.00	0	0	0.004
v150_d10	**371.00**	371.00	0	0	0.028
v150_d20	619.00	595.49	3.95	2,491	-
v150_d30	575.00	486.09	18.29	203	-
v150_d50	440.00	282.78*	55.61	2	-
v150_d70	∼	∼	∼	∼	∼
v200_d5	**163.00**	163.00	0	0	0.014
v200_d10	**680.00**	680.00	0	0	0.112
v200_d20	979.00	901.11	8.64	600	-
v200_d30	921.00	726.77	26.72	11	-
v200_d50	800.00	468.85*	70.62	1	-
v200_d70	∼	∼	∼	∼	∼

Table D.17: Branch-and-Price. Performance Metrics.

Instance	Deepest Depth	Average Depth	Branching Count	Pruned by bound	Pruned by integrality	Pruned by infeasibility
v30_d10	0	0	0	0	0	0
v30_d20	0	0	0	0	0	0
v30_d30	0	0	0	0	0	0
v30_d50	8	4.18	34	1	0	0
v30_d70	38	21.16	3,218	0	0	0
v50_d5	0	0	0	0	0	0
v50_d10	0	0	0	0	0	0
v50_d20	0	0	0	0	0	0
v50_d30	5	3.00	10	2	0	0
v50_d50	36	16.28	3,670	0	0	0
v50_d70	23	10.97	656	0	0	0
v70_d5	0	0	0	0	0	0
v70_d10	0	0	0	0	0	0
v70_d20	1	1.00	2	0	0	0
v70_d30	38	18.71	8,140	0	0	0
v70_d50	28	14.13	1,294	0	0	0
v70_d70	8	4.38	42	0	0	0
v100_d5	0	0	0	0	0	0
v100_d10	0	0	0	0	0	0
v100_d20	16	7.61	194	0	0	0
v100_d30	36	15.23	4,028	0	0	0
v100_d50	15	7.37	134	0	0	0
v100_d70	0	0	0	0	0	0
v120_d5	0	0	0	0	0	0
v120_d10	0	0	0	0	0	0
v120_d20	55	26.19	6,642	0	0	0
v120_d30	25	12.54	1,954	0	0	0
v120_d50	6	3.50	16	0	0	0
v120_d70	~	~	~	~	~	~
v150_d5	0	0	0	0	0	0
v150_d10	0	0	0	0	0	0
v150_d20	36	18.61	4,980	0	0	0
v150_d30	14	7.88	404	0	0	0
v150_d50	1	1.00	2	0	0	0
v150_d70	~	~	~	~	~	~
v200_d5	0	0	0	0	0	0
v200_d10	0	0	0	0	0	0
v200_d20	27	13.52	1,198	0	0	0
v200_d30	8	5.10	20	0	0	0
v200_d50	0	0	0	0	0	0
v200_d70	~	~	~	~	~	~

Table D.18: Branch-and-Price. Enumeration Tree Statistics.

Instance	VCCP Solution Feasibility	ECCP Solution Feasibility	$\overline{v}(\mathcal{R}_1)$
v30_d10	Yes	Yes	**40.00**
v30_d20	Yes	Yes	**48.00**
v30_d30	Yes	Yes	**55.00**
v30_d50	Yes	Yes	**40.00**
v30_d70	Yes	Yes	27.00
v50_d5	Yes	Yes	**57.00**
v50_d10	Yes	Yes	**96.00**
v50_d20	Yes	Yes	**119.00**
v50_d30	Yes	Yes	**115.00**
v50_d50	Yes	Yes	77.00
v50_d70	Yes	Yes	54.00
v70_d5	Yes	Yes	**113.00**
v70_d10	Yes	Yes	**172.00**
v70_d20	Yes	Yes	**186.00**
v70_d30	Yes	Yes	178.00
v70_d50	Yes	Yes	136.00
v70_d70	Yes	Yes	82.00
v100_d5	Yes	Yes	**208.00**
v100_d10	Yes	Yes	**326.00**
v100_d20	Yes	Yes	**329.00**
v100_d30	Yes	Yes	303.00
v100_d50	Yes	Yes	226.00
v100_d70	Yes	Yes	171.00
v120_d5	Yes	Yes	**319.00**
v120_d10	Yes	Yes	**433.00**
v120_d20	Yes	Yes	441.00
v120_d30	Yes	Yes	407.00
v120_d50	Yes	Yes	318.00
v120_d70	$\sim$	$\sim$	$\sim$
v150_d5	Yes	Yes	**471.00**
v150_d10	Yes	Yes	**617.00**
v150_d20	Yes	Yes	623.00
v150_d30	Yes	Yes	575.00
v150_d50	Yes	Yes	440.00
v150_d70	$\sim$	$\sim$	$\sim$
v200_d5	Yes	Yes	**751.00**
v200_d10	Yes	Yes	**972.00**
v200_d20	Yes	Yes	979.00
v200_d30	Yes	Yes	921.00
v200_d50	Yes	Yes	800.00
v200_d70	$\sim$	$\sim$	$\sim$

Table D.19: Branch-and-Price. Solution Feasibility.