UNIVERSIDADE FEDERAL FLUMINENSE

JOSÉ ANGEL RIVEAUX MERINO

PROBLEMAS DE OTIMIZAÇÃO EM QUASE-CLIQUES

NITERÓI 2022

UNIVERSIDADE FEDERAL FLUMINENSE

JOSÉ ANGEL RIVEAUX MERINO

PROBLEMAS DE OTIMIZAÇÃO EM QUASE-CLIQUES

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Ciência da Computa-

Orientador: Celso da Cruz Carneiro Ribeiro

ção

NITERÓI 2022

Ficha catalográfica automática - SDC/BEE Gerada com informações fornecidas pelo autor

R621p Riveaux merino, José Angel Problemas de optimização em quase-cliques / José Angel Riveaux merino ; Celso da Cruz Carneiro Ribeiro, orientador. Niterói, 2022. 126 f. : il. Tese (doutorado)-Universidade Federal Fluminense, Niterói, 2022. DOI: http://dx.doi.org/10.22409/PGC.2022.d.07346489101 1. Quase-Clique de cardinalidade máxima. 2. Particionamento mínimo em quase-cliques. 3. Grafos. 4. Algoritmo genético tendencioso de chaves aleatória. 5. Produção intelectual. I. Carneiro Ribeiro, Celso da Cruz, orientador. II. Universidade Federal Fluminense. Instituto de Computação. III. Título.

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

José Angel Riveaux Merino

PROBLEMAS DE OTIMIZAÇÃO EM QUASE-CLIQUES

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação.

Área de concentração: Ciência da Computação.

Aprovada em Abril de 2022.

BANCA EXAMINADORA

Alorana

Prof. Celso da Cruz Carneiro Ribeiro - Orientador, UFF

Profa. Isabel C. M. Rosseti, UFF

ke h

Prof. Nelson Maculan Filho, COPPE/UFRJ

hand a he Prof. Rafael Augusto de Melo, UFBA

thiago J. Noronha

Prof. Thiago Ferreira de Noronha, UFMG

Verturon do Sonto Surgo

Prof. Uéverton dos Santos Souza, UFF

Niterói

2022

Resumo

A mineração de dados em redes estuda propriedades de sistemas complexos onde as entidades do sistema são representadas por vértices e os relacionamentos entre elas são representados por arestas. A análise de redes encontra aplicações relevantes por exemplo em pesquisas relacionadas com a propagação de pandemias como a Covid-19, nas colaborações tecnológicas na captura e no armazenamento de carbono, na análise da polarização de redes sociais, entre muitas outras.

A densidade de um grafo é uma forma de medir a coesão dos elementos de uma rede ou de subgrupos dentro da mesma. Quase-cliques, ou γ -cliques, são subconjuntos de vértices do grafo que induzem um subgrafo de densidade maior ou igual a um valor $\gamma \in (0, 1]$. Quase-cliques são uma generalização do conceito de clique: uma γ -clique é uma clique quando $\gamma = 1$.

Esta tese trata dois problemas de otimização combinatória relacionados com quasecliques: o problema da quase-clique de cardinalidade máxima e o problema de particionamento mínimo de um grafo em quase-cliques. Algoritmos exatos e heurísticas foram desenvolvidos e apresentados para os problemas. É estudada a complexidade do problema de particionamento de grafos em quase-cliques, mostrando-se que até mesmo a versão de decisão do problema de bipartição em quase-cliques é um problema NP-completo.

Como contribuição adicional, buscou-se uma forma de melhorar o desempenho do algoritmo genético tendencioso de chaves aleatória (BRKGA). Mostrou-se como o uso da estratégia de reconexão de caminhos no espaço das chaves aleatórias pode ser explorado com sucesso como uma estratégia de cruzamentos progressivos que melhora o desempenho do BRKGA.

Palavras-chave: Quase-clique, clique, grafos, densidade, otimização combinatória, particionamento.

Abstract

Data mining in networks studies properties of complex systems where the entities are represented by vertices and the relationships between them are represented by edges. The analysis of network has relevant applications, for example in research related to the spread of pandemics such as Covid-19, in technological collaborations, in carbon capture and storage and in the analysis of the polarization of social networks, among many others.

The density of a graph is used to measuring the cohesion of the elements or subgroups of elements of the network. Quasi-cliques, or gamma-cliques, are subsets of vertices of a graph that induce a subgraph of density greater than or equal to a value $\gamma \in (0.1)$. Quasi-cliques are a generalization of the clique concept: a gamma-clique is a clique when $\gamma = 1$.

This thesis deals with two combinatorial optimization problems related to quasicliques: the maximum quasi-clique problem and the minimum quasi-clique partitioning problem. Exact algorithms and heuristics were developed and presented for both problems. The complexity of the graph partitioning problem in quasi-cliques was also studied, showing that even the decision version of the quasi-cliques bipartitioning problem is an NP-complete problem.

As an additional contribution, the biased random-key genetic algorithm (BRKGA) metaheuristic was used as a component of the proposed matheuristic for the maximum quasi-clique problem. Also, it was shown how the use of the path-relinking strategy in the random key space can be successfully exploited as a progressive crossover strategy that improves the performance of BRKGA.

Keywords: Quasi-clique, clique, graph, density, combinatorial optimization problems, partitioning.

Lista de Figuras

3.1	Evolução da população no algoritmo BRKGA	 11
5.1	Evolução do BRKGA com dois operadores de cruzamento	 17

Lista de Abreviaturas e Siglas

$\gamma\text{-}\mathrm{MQCP}$:	problema da γ -clique de cardinalidade máxima;
γ -MQCPP	:	problema de particionamento mínimo em γ -cliques;
$\gamma\text{-}\text{QCPP}$:	problema de particionamento em γ -cliques;
QCBP	:	problema de bi-particionamento em quasi-clique;
MCP	:	problema de corte máximo;
BRKGA	:	algoritmo genético tendencioso com chaves aleatória;
MIP	:	Programação Misto-Inteira (Mix-Integer Programming)
DLS	:	problemas de escalonamento de tarefas divisíveis (Divisible Load Scheduling)
DLS-SR	:	problema de escalonamento em uma única etapa de distribuição de tarefas ;
DLS-MR	:	problema de escalonamento com varias iterações.

Sumário

-	Intr	ntrodução				
	1.1	Definições	1			
	1.2	Métodos de solução	2			
	1.3	Aplicações dos problemas de otimização em quase-cliques	3			
	1.4	Estado da arte	4			
	1.5	Principais contribuições	5			
	1.6	Estrutura da tese	6			
	1.7	Lista de publicações	6			
2	Um máx	algoritmo exato para o problema de quasi-clique de cardinalidade ima	8			
3	Um	a matheurística baseada em BRKGA para o problema de quasi-				
	cnq	a de condinalidade mérrime com busce legal evete	10			
		le de cardinalidade máxima com busca local exata	10			
	3.1	le de cardinalidade máxima com busca local exata Algoritmos genéticos tendenciosos com chaves aleatórias	10 10			
	3.1 3.2	le de cardinalidade máxima com busca local exata Algoritmos genéticos tendenciosos com chaves aleatórias Matheurística baseada em BRKGA para o problema da quase-clique máxima	101012			
4	3.1 3.2 Pro	ae de cardinalidade máxima com busca local exata Algoritmos genéticos tendenciosos com chaves aleatórias Matheurística baseada em BRKGA para o problema da quase-clique máxima olema de particionamento mínimo em quase-cliques	 10 10 12 14 			
4	3.1 3.2 Pro BRI	ae de cardinalidade máxima com busca local exata Algoritmos genéticos tendenciosos com chaves aleatórias Matheurística baseada em BRKGA para o problema da quase-clique máxima olema de particionamento mínimo em quase-cliques KGA com reconexão por caminhos no espaço das chaves e aplicações	 10 12 14 16 			
4 5 6	3.1 3.2 Pro BRI Con	ae de cardinalidade máxima com busca local exata Algoritmos genéticos tendenciosos com chaves aleatórias Matheurística baseada em BRKGA para o problema da quase-clique máxima olema de particionamento mínimo em quase-cliques KGA com reconexão por caminhos no espaço das chaves e aplicações clusões	 10 12 14 16 19 			

Apêndice A – An exact algorithm for the maximum quasi-clique problem 26

Apêndice B – A BRKGA-based matheuristic for the maximum quasi-clique	
problem with an exact local search strategy	58
Apêndice C – The minimum quasi-clique partitioning problem: Comple- xity, formulations, and a greedy randomized heuristic	82
Apêndice D – Biased random-key genetic algorithms using path-relinking	
as a progressive crossover strategy	107

Capítulo 1

Introdução

Neste capítulo se expõem os principais problemas de otimização combinatória tratados nesta tese e os métodos de solução propostos. Apresenta-se um resumo das aplicações dos problemas tratados e das pesquisas relacionadas. Também são apresentadas as contribuições da tese.

1.1 Definições

Seja G = (V, E) um grafo definido por um conjunto de vértices V e um conjunto de arestas $E \subseteq V \times V$. Diz-se que G é um grafo completo se existe uma aresta em E para cada par de vértices de V. O grafo G' = (V', E') é um subgrafo de G se $V' \subseteq V$ e $E' \subseteq E$, o que é denotado como $G' \subseteq G$. O grafo G(V') induzido em G por $V' \subseteq V$ é aquele com o conjunto de vértices V' e com o subconjunto de arestas de E que tem as duas extremidades em V'. Para qualquer $V' \subseteq V$, o subconjunto $E(V') \subseteq E$ é formado por todas as arestas de E com ambas extremidades em V' (i.e., E(V') é o conjunto de arestas do grafo induzido em G por V').

O grau de um vértice $v \in V$, denotado como $deg_G(v)$, é o número de vértices adjacentes a v em G. Para todo $v \in V$ e para todo $V' \subseteq V$, denota-se por $deg_G(v, V')$ o número de vértices de V' que são adjacentes a v em G.

A densidade do grafo G é dada por $dens(G) = |E|/(|V| \times (|V| - 1)/2)$. Nota-se que a densidade de um grafo completo é igual a um.

Um subconjunto $C \subseteq V$ é uma clique de G se o subgrafo G(C) induzido em G por C é completo. Dado um grafo G = (V, E), o problema da clique máxima consiste em encontrar a clique de cardinalidade máxima de G. Este problema está entre os primeiros

problemas provados como NP-difíceis em [18].

Seja $\gamma, \lambda \in (0, 1]$ Pattillo, Youssef e Butenko [30] apresentam duas definições de quasecliques:

Definição 1 γ -clique: $C \subseteq V$ é uma γ -clique se dens $(G(C)) \geq \gamma$.

Definição 2 (λ, γ) -clique: $C \subseteq V$ é uma (λ, γ) -clique se $dens(G(C)) \geq \gamma$ e para todo vértice $v \in C$, $deg_G(v, C) \geq \lambda(|C| - 1)$.

Seguindo a definição de Resende e Ribeiro [35], um problema de otimização combinatória pode ser definido como: otimizarf(S) sujeito a: $S \in F$. F é um conjunto finito ou infinito e contável de soluções viáveis do problema e f é uma função objetivo que associa a cada solução $S \in F$ um custo ou valor real f(S). No caso dos problemas de minimização se procura a solução $S^* \in F$ tal que $f(S^*) \leq f(S), \forall (S) \in F$. No caso de maximização se procura a solução $S^* \in F$ tal que $f(S^*) \geq f(S), \forall (S) \in F$. A solução S^* é um ótimo global.

Este trabalho foca sua atenção no estudo de dois problemas de otimização combinatória relacionados com γ -cliques:

Quase-clique de cardinalidade máxima

Dado um grafo G = (V, E), o problema da γ -clique de cardinalidade máxima (γ -MQCP) consiste em encontrar o subconjunto C^* de vértices de V de maior cardinalidade, tal que a densidade do grafo induzido em G por C^* seja maior ou igual a γ .

Particionamento mínimo em quase-cliques

Dado um grafo G = (V, E), um particionamento $P = V_1, V_2, \ldots, V_k$ dos nós de G é tal que $V_1 \cup V_2 \cup \ldots \cup V_k = V$ e $V_i \cap V_j = \emptyset$ para todo $i, j = 1, \ldots, k$ com $i \neq j$. Logo, dado $\gamma \in (0, 1]$, o problema de *particionamento mínimo em* γ -cliques (γ -MQCPP) consiste em encontrar o número mínimo de subconjuntos do grafo G tal que todos tenham densidade maior ou igual a γ e que formem uma partição do conjunto V de nós.

1.2 Métodos de solução

Métodos exatos para solucionar problemas de otimização são aqueles que garantem encontrar, em tempo finito, um **ótimo global** do problema e provar a sua otimalidade, ou mostrar que não o problema não admite solução viável [35]. Heurísticas ou métodos aproximados são aqueles que encontram uma solução viável do problema, mas não necessariamente uma solução ótima. Metaheurísticas são procedimentos gerais de alto nível que coordenam heurísticas simples e regras para encontrar soluções de boa qualidade para problemas de otimização computacionalmente difíceis [35].

Matheurísticas, ou heurísticas baseadas em modelos, são procedimentos que procuram combinar técnicas de programação matemática com meta(heurísticas) para: (i) melhorar a eficiência em tempo computacional das abordagens exatas por programação matemática a partir de soluções encontradas por metaheurísticas ou (ii) melhorar a qualidade da soluções obtidas por metaheurísticas ao explorar a formulação do modelo de programação matemática [37].

Neste trabalho propõem-se métodos exatos, heurísticas e matheurísticas para solucionar os problemas definidos anteriormente.

1.3 Aplicações dos problemas de otimização em quasecliques

Aplicações típicas dos problemas de otimização em quase-cliques podem ser encontradas em mineração de dados em redes, onde são estudadas propriedades de sistemas complexos nos quais as entidades do sistema são representadas por vértices e os relacionamentos entre elas são representados por arestas [4]. Como exemplo podem ser citadas as redes sociais, onde os usuários (com seus perfis) constituem as entidades. Uma das medidas mais frequentemente usadas na análise das redes é sua coesão, que permite avaliar o quão forte é a conexão entre seus elementos.

A coesão entre os elementos pode ser avaliada a partir da densidade da rede ou subrede, sendo o clique um grupo coeso "perfeito". Na realidade, uma clique pode ser uma estrutura forte demais para definir a coesão já que, especialmente em grafos grandes e esparsos, (i) pode não ser necessário que todas as entidades do grupo estejam conectadas entre si ou (ii) a perda de informações ou dados sobre as entidades pode fazer com que a relações não sejam refletidas na rede. Desta forma, pode ser mais interessante modelar grupos como k-cores (subgrafos maximais com grau maior ou igual a k) ou relaxações de clique tais como s-plexes (subgrafos nos quais todo vértice não é adjacente a pelo menos s - 1 dos vértices restantes), s-clubs (conjuntos de vértices que induzem um subgrafo de diâmetro maior ou igual a s) ou quase-cliques [4, 20, 48]. Pattillo, Youssef e Butenko [30] mostraram a importância dos problemas de otimização com as diferentes relaxações

de cliques, entre elas as quase-cliques, para estudar a coesão de subgrupos em redes, i.e. a conectividade do subconjunto de vértices do grafo. No caso das redes sociais, esta medida pode ser usada na detecção de comunidades, para identificar redes de criminosos, para estudar a polarização nas redes sociais ou a influencia do comportamento social na propagação de epidemias, entre outros.

A polarização é um fenômeno no qual uma rede (por exemplo, uma rede social) está composta por subgrupos coesos com uma débil conexão intergrupos. Garcia et al. [11] fizeram um análise da polarização na plataforma online politnetz.ch [6], onde políticos suíços interagiam criando links de apoio, comentários e curtidas. Os autores usaram algoritmos de particionamento de grafos, entre outras técnicas de análise de redes, para o estudo da polarização na rede de apoio e de interação exclusiva entre políticos.

Marqués et al [24] procuram subgrupos coesos analisando as redes sociais em residências universitárias, facilitando a compreensão do comportamento social dos estudantes e avaliando quais tipos de estruturas sociais estudantis apresentam melhor ou pior desempenho em relação ao contágio do COVID-19. Yin, Gu e Zhang [51] também procuram subgrupos coesos em uma rede global de cooperação de patentes, junto com outras técnicas de análise da redes sociais, para medir as colaborações tecnológicas na captura e armazenamento de carbono estudando uma rede global de cooperação de patentes.

Outras aplicações podem ser também encontradas em outras áreas tais como: telecomunicações [2], biologia [43], bioinformática [17] ou computação quântica [49].

1.4 Estado da arte

Problemas relacionados especificamente com a obtenção da quase-clique de cardinalidade máxima, ou pelo menos maximais, vêm chamando a atenção de vários pesquisadores. Abello, Resende e Sudarsky [2] propuseram uma solução que aplica preprocessamento e GRASP para obter quase-cliques em grafos esparsos grandes. Tsourakakis et al. [45] definiram o problema de obtenção do que eles denominam um *quase-clique ótimo*, que corresponde a determinar o subconjunto de vértices $S \subseteq V$ que maximiza o valor da função $f_{\gamma}(S) = |E(S)| - \gamma {|S| \choose 2}$. Eles propuseram um algoritmo adaptativo de aproximação e uma heurística de busca local para obter os quase-clique ótimos. Pattillo et al. [29] também estudaram o problema de γ -MQCP. Os autores mostraram que o problema é NP-difícil para qualquer $0 < \gamma < 1$, propuseram formulações para o calculo dos limites superiores e apresentaram formulações de programação inteira. Veremyev et al. [47] propuseram novas formulações de programação inteira para o γ -MQCP e as compararam téorica e computacionalmente com as apresentadas em [29]. Pinto et al. [32] elaboraram um algoritmo genético tendencioso de chaves aleatórias (BRKGA) para γ -MQCP. Zhou, Benlic e Wu [52] descreveram um algoritmo memético para o problema, que utiliza busca tabu para melhorar a solução localmente e aprendizado baseado em oposição (OBL) para melhorar o processo de busca. Marinelli, Pizzuti e Rossi [23] propuseram uma nova formulação de programação inteira para γ -MQCP com um número exponencial de variáveis que fornece um limite mais forte da relaxação linear do que as formulações existentes.

O problema γ -MQCPP de particionamento em γ -cliques ainda não foi estudado na literatura, apesar de suas aplicações. O único trabalho conhecido que considera o problema de particionamento mínimo de um grafo em quase-cliques é Basu et al. [4]. Os autores propuseram uma abordagem baseada em teoria de jogos para o particionamento do grafo em (λ, γ) -cliques associados a comunidades. Porém, o problema não é analisado sob o ponto de vista de otimização. Em vez disso, as partições são avaliadas de acordo com as medidas de qualidade da rede em relação às comunidades encontradas.

1.5 Principais contribuições

As principais contribuições desta tese são enumeradas a seguir:

1. A primeira contribução desta tese é um algoritmo exato enumerativo para γ -MQCP, baseado na propriedade quasi-hereditária dos γ -cliques (ver apêndice A) [39]. Se propõe uma estratégia para a poda da árvore de busca, que leva a uma nova formulação para o calculo de um limite superior para o problema. Os resultados númericos mostram que os limites obtidos são menores (i.e., melhores) ou iguais aos limites analíticos propostos por Pattillo et al. [29]. O algoritmo proposto é competitivo quando comparado com as melhores formulações de programação inteira existentes até o momento da pesquisa e com o algoritmo enumerativo de [28].

2. Foi proposta uma matheurística para γ -MQCP baseada na hibridação do algoritmo genético tendencioso com chaves aleatórias (BRKGA) com uma estrategia de busca local exata (ver apêndice B) [31]. A nova abordagem é comparada com o BRKGA puro de Pinto et al. [32], que era a melhor heurística existente na literatura no momento de desenvolvimento da pesquisa. Resultados computacionais mostram que o BRKGA hibrido obteve melhores resultados que o original.

3. Foi introduzido o problema de particionamento mínimo em quase-cliques (γ -MQCPP)

e se mostrou que o mesmo é NP-difícil, mesmo quando a solução ótima é igual a dois, i.e., quando a instancia do grafo admite um particionamento em dois quase-cliques (ver apêndice C). Também se apresentam e comparam computacionalmente quatro formulações de programação inteira (com um número polinomial de variáveis e restrições) para o problema. Propõe-se também uma heurística gulosa e aleatória com inicialização múltipla, para encontrar uma solução inicial viável para as formulações [25].

4. Paralelamente ao desenvolvimento da matheurística para o γ -MQCP, desenvolveu-se a extensão da ideia de reconexão por caminhos (ou path-relinking) [35, 38] como operação de cruzamento progressivo dentro da metaheurística BRKGA (ver apêndice D) [40]. Apesar de reconexão por caminhos ter sido aplicada com êxito na hibridação com diversas metahéuristicas como busca tabu [12], scatter search [13, 14, 15, 36] e GRASP [33, 35], esta técnica tem sido pouco aplicada em algoritmos genéticos [41] e nos algoritmos genéticos tendenciosos com chaves aleatórias [3]. As primeiras aplicações e resultados numéricos foram obtidos sobre instâncias do problema de escalonamento de tarefas divisíveis (divisible load scheduling) [8, 9, 10].

1.6 Estrutura da tese

A presente tese queda estruturada em quatro capítulos, do capítulo 2 ao 5, como resumos estendidos do conteúdo do apêndice correspondente.

No Capítulo 2 introduzido o algoritmo exato para γ -MQCP.

O capítulo 3 apresenta a matheurística BRKGA-LSQC
lique para γ -MQCP.

O Capítulo 4 aborda a complexidade do γ -MQCPP e as diferentes formulações de programação inteira propostas para este problema.

Finalmente, no Capítulo 5 mostra-se como pode ser aplicada a ideia de *path-relinking* como operação de cruzamento dentro do BRKGA.

Para finalizar, no Capítulo 6 são apresentadas as conclusões desta tese.

1.7 Lista de publicações

 Ribeiro, C. C.; Riveaux, J. A. "An exact algorithm for the maximum quase-clique problem". International Transactions in Operational Research 26 (2019), 2199-2229.

- Pinto, B. Q.; Ribeiro, C. C.; Riveaux, J.-A.; Rosseti, I. "A BRKGA-based matheuristic for the maximum quase-clique problem with an exact local search strategy". *RAIRO Operations Research* 55 (2021), 741–763.
- Melo, R. A.; Ribeiro, C. C.; Riveaux, J. A. "The minimum quase-clique partitioning problem: Complexity, formulations, and a greedy randomized heuristic". Submetido para publicação.
- Ribeiro, C. C.; Riveaux, J. A.; Brandão, J. S. "Biased random-key genetic algorithms using path-relinkingas a progressive crossover strategy", *Proceedings of the* 5th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence (ISMSI 2021), Association for Computing Machinery, páginas 28-36.

Capítulo 2

Um algoritmo exato para o problema de quasi-clique de cardinalidade máxima

O problema da γ -clique de cardinalidade máxima (γ -MQCP) foi provado ser NP-difícil por Patillo et al. [29]. No mesmo trabalho prova-se que a propriedade de um grafo ter densidade no mínimo γ é quase-hereditária, ou seja, toda γ -clique de cardinalidade kcontem pelo menos um subconjunto de vértices de cardinalidade k-1 que também é uma γ -clique. Os autores apresentaram duas formulações para o cálculo de um limite superior da cardinalidade da γ -clique máxima.

As formulações MIP existentes até o momento do desenvolvimento deste trabalho foram propostas em [29, 47]. Aquelas apresentadas em [47] mostraram melhor desempenho em instancias pouco densas. As duas melhores, dentre todas as existentes, foram selecionadas para avaliar a eficiência do algoritmo exato *QClique* [39] proposto nesta tese. O algoritmo exato enumerativo baseado na busca em profundidade, proposto em [28], também foi usado para comparações com o algoritmo *QClique* nos experimentos computacionais.

O algoritmo enumerativo *QClique* proposto por Ribeiro e Riveaux [39] e reproduzido no Apéndice A se baseia fundamentalmente na propriedade quase-hereditária da densidade do grafo. Esta propriedade garante que:

Proposição 1 Para qualquer um γ -clique C de cardinalidade k em G, existe uma permutação Π dos vértices de C tal que, para todo $i = 1, \ldots, k$, o ultimo vértice v do prefixo Π_i , de tamanho i, de Π tem grau mínimo no grafo induzido em G pelos vértices de Π_i .

9

Etiquetando os vértices de G com 1, 2, ..., |V| e considerando a proposição 1, o algoritmo *Backtracking*, proposto no artigo, garante que todas as soluções viáveis sejam enumeradas sem repetições.

O algoritmo enumerativo é acelerado a partir da poda da árvore de busca. Em cada nó se busca encontrar uma solução de tamanho k que melhore a melhor solução encontrada até o momento, que possui tamanho k - 1. O nó atual possui um prefixo Π_i de tamanho i < k. Para todo $j = 1, \ldots, k - i$, e para qualquer Π_{i+j} com prefixo Π_i , não conhecido, são calculados os limites superiores do número de arestas em G entre os vértices de Π_i e os vértices de Π_{i+j} que não pertencem a Π_i e do número de arestas em G entre os vértices de Π_{i+j} que não pertencem a Π_i . Se, para algum $j = 1, \ldots, k - i$, a soma dos limites superiores antes calculado com o numero de arestas do grafo induzido em G pelos vértices de Π_i não for maior ou igual ao número de arestas de um $\gamma - clique$ de tamanho i + j, então o nó atual pode ser podado da árvore de enumeração.

A partir da estratégia usada na poda, pode-se concluir que o valor da solução ótima é menor ou igual a $newUB = \max\{k : k = 1, ..., |V| \in \sum_{v \in \Pi''} \deg_G(v)/2 \ge \gamma \cdot k(k-1)/2\},$ onde Π'' são os k vértices de maior grau em G. Nos experimentos apresentados no artigo mostra-se como este novo limite superior supera os apresentados por Patillo et. al [29].

O algoritmo QClique consegue bons resultados, sobretudo em instancias grandes de baixa densidade quando comparado com os resultados das melhores formulações apresentadas em [29, 47] e com o algoritmo enumerativo de [28]. Por outro lado, mostra-se também que a poda reduz consideravelmente o numero de nós da árvore de busca, que se traduz em uma melhoria considerável no tempo para algumas instancias como na instância email com $\gamma = 0.75$ onde de 76.742.311 nós, apenas 7.810.753 são visitados e o corte é aplicado 7.042.802 vezes.

Capítulo 3

Uma matheurística baseada em BRKGA para o problema de quasiclique de cardinalidade máxima com busca local exata

3.1 Algoritmos genéticos tendenciosos com chaves aleatórias

Um algoritmo genético tendencioso com chaves aleatórias (*Biased Random Key Genetic Algorithm*, ou simplesmente BRKGA) evolui uma população de cromossomos representados por vetores de números reais, eventualmente gerados de forma aleatória, denominados chaves. Cada um dos componentes (ou alelo) das chaves é um número real no intervalo (0, 1]. Os cromossomos são decodificados por um algoritmo determinístico que associa a cada chave uma solução viável do problema de otimização a ser resolvido, cuja adaptação ou valor da função objetivo pode ser calculada.

Uma das caraterísticas fundamentais dos BRKGA é que cada nova solução gerada por cruzamento é obtida pela combinação de uma solução selecionada aleatoriamente do subconjunto de soluções elite da população atual com outra que não pertence à população de elite. Não apenas a seleção é tendenciosa, porque um dos pais é sempre da população elite, mas também este é o que tem maior probabilidade de passar suas caraterísticas para sua descendência.

Outra caraterística fundamental é que toda solução obtida por cruzamento sempre gera uma solução viável, já que o decodificador sempre constrói soluções viáveis partindo de qualquer sequencia de chaves aleatórias. Para a operação de cruzamento dos pais é usada a função de cruzamento paramétrica e uniforme de Spears e de Jong [42], onde cada descendente herda com maior probabilidade cada uma de suas chaves do pai com melhor valor da função objetivo. Por outro lado, em vez da operação de mutação que é usualmente usada nos algoritmos genéticos tradicionais, são introduzidas como soluções mutantes novos cromossomos gerados aleatoriamente. O objetivo de gerar novas soluções é o mesmo da operação de mutação tradicional, que é o de diversificar a população e ajudar o algoritmo a escapar de ótimos locais, ver [8, 9, 16, 26, 34].

A população atual é particionada em dois subconjuntos em cada geração: TOP e REST. O subconjunto TOP contém as soluções de elite e é formado pelas melhores soluções da população. O subconjunto REST é dividido em dois subconjuntos: MID e BOT, onde o subconjunto BOT é formado pelas piores soluções da atual geração. O tamanho da população é dado por |TOP| + |REST|.

A evolução de uma geração para outra é ilustrada na Figura 3.1 (ver Pinto et al. [32]). Primeiramente, as soluções em TOP são copiadas para a próxima geração. Os elementos em BOT são substituídos por soluções mutantes geradas aleatoriamente. As restantes |MID| = |REST| - |BOT| soluções serão obtidas pela combinação (cruzamento) de um pai escolhido aleatoriamente de TOP e outro de REST, ambos da geração atual. O cruzamento é feito no espaço das chaves e o decodificador é aplicado para gerar uma solução viável. Esta seria a principal diferença entre o algoritmo BRKGA e o algoritmo genético de chaves aleatórias (RKGA) de Bean [5], onde ambos pais são escolhidos aleatoriamente entre toda a população.



Figura 3.1: Evolução da população no algoritmo BRKGA.

3.2 Matheurística baseada em BRKGA para o problema da quase-clique máxima

Duas variantes do BRKGA foram desenvolvidas para o γ -MQCP em [32], cada uma partindo de um dos decodificadores DECODER-HCB e DECODER-IG^{*}. O algoritmo evolui uma população formada por vetores de valores reais no intervalo (0, 1], onde cada componente do vetor é associada a um dos nós do grafo G. Ambas variantes do BRKGA foram implementadas em C++ usando a biblioteca brkgaAPI desenvolvida por Toso e Resende [44]. O uso desta biblioteca permite que o usuário somente precise implementar o decodificador, já que esta é a única parte do algoritmo que depende do problema.

O decodificador DECODER-HCB está baseado no algoritmo construtivo do GRASP apresentado por Abello, Resende e Sudarsky al. [2]. As chaves são representadas por vetores de tamanho |V|. O decodificador DECODER-IG* é uma extensão do primeiro. Neste caso, as chaves são representadas por vetores de tamanho $2 \cdot |V|$, onde os primeiros |V| elementos são usados no processo de construção da solução. Os demais |V| elementos são usados depois para destruir parte da solução, eliminando uma fração δ de seus vértices com um algoritmo guloso onde o parâmetro β determina sua voracidade, controlando o tamanho da lista restrita de candidatos. Logo, volta-se a reconstruir a solução usando novamente o algoritmo DECODER-HCB. O processo de destruição e reconstrução é repetido ate que a solução não possa mais ser melhorada. Os decodificadores são detalhados em [31, 32] e no Apêndice B.

A heurística do BRKGA-IG^{*} mostrou melhor desempenho em comparação com o algoritmo guloso iterativo otimizado com estratégia de reinicio (RIG^{*}), apresentado por Oliveira, Plastino e Ribeiro [27]. Ele também foi comparado com os algoritmos AlgF3 e AlgF4 de Veremyev et al. [47], usado como heurísticas com limite no seu tempo de execução. O BRKGA-IG^{*} aplicado a grafos pouco densos também obteve melhores resultados que as formulações MIP, obtendo valores alvos em menores tempos de execução.

Como parte desta tese, foi proposta uma melhoria da heurística BRKGA-IG^{*} usando um algoritmo exato para o γ -MQCP. A idéia consiste em substituir a reconstrução feita com o DECODER-HCB pelo algoritmo exato para o γ -MQCP, que resulta na *matheurística* BRKGA-LSQClique. A descrição detalhada do algoritmo encontra-se no artigo [31] no Apêndice B.

Para 150 combinações de instancias e valores de γ , os algoritmos BRKGA-IG^{*} e BRKGA-LSQClique foram executados 30 vezes, até um valor alvo ser alcançado dentro de um tempo limite. Os resultados forem comparados em termos do número de vezes que o valor alvo foi alcançado e do tempo médio para alcançá-lo. Tendo em vista estes critérios, se concluiu que o algoritmo BRKGA-IG* se comportou melhor para 94.44% das 54 combinações quando $\gamma = 0.999$, enquanto que BRKGA-LSQClique foi consideravelmente melhor para valores menores de γ , chegando a ser melhor em 100% das combinações com $\gamma = 0.95$, em 91.67% das combinações com $\gamma = 0.90$ e em 95.65% das combinações com $\gamma \leq 0.80$. Resultados detalhados são apresentados no artigo [31].

Capítulo 4

Problema de particionamento mínimo em quase-cliques

Embora o problema de particionamento em quase-cliques seja de interesse, por exemplo na área de agrupamentos em redes (*network clustering*) [30, 48], este problema tem sido pouco explorado e são desconhecidos, até o momento, artigos que proponham algoritmos exatos ou aproximados para sua solução.

Melo, Ribeiro e Riveaux demonstram no artigo [25] (apresentado no apêndice C) que o problema γ -MQCPP é NP-difícil, inclusive para o caso mais restrito onde a solução ótima é igual a dois. Para efeito dessa demonstração se considera o problema de decisão associado definido como:

Problema de particionamento em quase-cliques (γ -QCPP)

Instancia: Grafo G = (V, E), um valor $\gamma \in (0, 1]$, e um inteiro $\mathcal{K} \ge 2$.

Pergunta: Existe um particionamento de $G \text{ em } \mathcal{K} \gamma$ -cliques?

Observe-se que para $\gamma = 1$, o problema γ -MQCPP corresponde ao problema de particionamento mínimo em cliques (CPP), que é NP-completo apenas quando se busca uma partição em mais de dois cliques [18]. Em [25], como parte desta tese, se mostra como o **problema de corte máximo sem pesos (MCP)** se reduz ao problema de particionamento em quase-cliques com $\mathcal{K} = 2$, denominado **problema de bi-particionamento em quase-clique (QCBP)**. O MCP e definido como:

Problema de corte máximo (MCP) Instancia: Grafo G = (V, E) e inteiro L. Pergunta: Existe $P \subset V$ tal que $|E(P, \overline{P})| \ge L$? Dada uma solução qualquer do **QCBP**, representada pela partição (S, \overline{S}) , com $S \subset V$ e $S \neq \emptyset$, as densidades dos grafos G(S) e $G(\overline{S})$ induzidos em G por S e \overline{S} , respetivamente, podem ser computada em tempo linear O(|V| + |E|). Logo, tendo em conta também a redução polinomial de **MCP** em **QCBP**, se demostra que o problema γ -QCPP é NPcompleto inclusive para $\mathcal{K} = 2$ e como consequência o problema γ -MQCPP é NP-difícil.

No artigo foram propostas quatro formulações por programação inteira: a formulação padrão (STD), a formulação padrão usando a descomposição por tamanho dos quasecliques (STDs), a formulação por representantes (REP) e a formulação por representantes usando a descomposição por tamanho dos quase-cliques (REPs). Propõe-se uma heurística multi-partida gulosa randomizada que fornece uma solução viável inicial para as formulações propostas. Esta heurística está baseada na heurística construtiva HCB [32] para o problema da quase-clique máxima (que é uma heurística gulosa randomizada onde o critério guloso leva em consideração a diferença de potencial introduzida por Abello, Pardalos e Resende al. [1].

Os experimentos computacionais mostraram que a formulações REP e REPs tiveram melhor desempenho. A formulação REP provou a otimalidade da solução para um número maior de instancias, mas REPs obteve os melhores valores de solução. Os resultados indicaram que instancias com valores de γ entre 0.40 e 0.70 tendem a ser mais difíceis de serem resolvidas pelas formulações. Os experimentos mostraram que a heurística proposta pode obter, em pouco tempo computacional, soluções viáveis de qualidade razoável a serem oferecidas as formulações.

Capítulo 5

BRKGA com reconexão por caminhos no espaço das chaves e aplicações

Paralelamente ao estudo dos problemas de otimização envolvendo quase-cliques, foi desenvolvida uma proposta para a implementação de reconexão de caminhos em BRKGAs, atuando no espaço das chaves. A reconexão de caminhos é uma estratégia para intensificação da busca que explora caminhos que conectam soluções de elite (i.e., de alta qualidade), em problemas de otimização combinatória [35]. Como um grande aprimoramento dos métodos de busca heurístico, sua hibridização com outras metaheurísticas tem levado a melhorias significativas na qualidade da solução e nos tempos de execução.

No contexto desta proposta, foram implementadas duas novas estrategias. A primeira consiste no uso de mais de um operador de cruzamento. Nos testes executados, foram considerados apenas dois operadores. Um parâmetro $pl \in [0, 1]$ determina a probabilidade do primeiro operador ser aplicado, enquanto 1 - pl é a probabilidade do segundo operador ser aplicado. Como consequência, o conjunto MID na próxima geração será formado por aproximadamente $pl \cdot |MID|$ descendentes gerados com o primeiro operador e $(1 - pl) \cdot |MID|$ gerados com o segundo operador, como ilustrado na Figura 5.1. A seleção dos pais pode permanecer igual ou mudar, dependendo da aplicação em si.

A segunda estrategia é a aplicação da reconexão de caminhos no espaço das chaves aleatórias como um operador de cruzamentos progressivos dentro do BRKGA. Sua implementação é detalhada no Apêndice D [40].

Para os experimentos computacionais, foram considerados dois problemas de escalonamento de tarefas divisíveis (*Divisible Load Scheduling*, ou DLS). Uma tarefa divisível



Figura 5.1: Evolução do BRKGA com dois operadores de cruzamento.

é uma quantidade $W \geq 0$ de trabalho computacional que pode ser arbitrariamente dividida e distribuída entre diferentes processadores para ser processada em paralelo. Os processadores são organizados em uma topologia de estrela e a carga é armazenada em um processador mestre central. O processador mestre divide a carga em fragmentos de tamanhos diferentes e transmite cada uma aos outros processadores. Cada processador pode somente iniciar o processamento depois de ter recebido completamente o seu fragmento de carga. Os processadores são heterogêneos em termos de poder de processamento, velocidade de comunicação e tempo de configuração para iniciar a comunicação com o processador mestre. Não é preciso usar todos os processadores disponíveis. Logo, independentemente de como é dividida a carga, a opção de (i) quais processadores são usados e (ii) a ordem em que os fragmentos são enviados influencia o tempo total de processamento da carga. O problema de escalonamento de tarefas (DLS) foi introduzido em [10], motivado por uma aplicação em sensores inteligentes de redes. Aplicações de DLS surgem em numerosos problemas de computação científica, como a busca paralela em bases de dados [7], processamento de imagens em paralelo [19], codificação de vídeos em paralelos [21, 46], processamento de grandes arquivos distribuídos [50] e programação de tarefas em computação em nuvem [22], entre outras.

Brandão et al. [8] trataram o problema de escalonamento em uma única etapa de distribuição de tarefas (DLS-SR), na qual cada processador ativo recebe um único fragmento da carga. Em [9] foi tratada a versão do problema com varias iterações (DLS-MR), onde diversos fragmentos de carga são enviados a cada processador que participa da computação, por meio de iterações sucessivas. A definição formal dos problemas pode ser vista nos artigos originais [8, 9]. Os experimentos computacionais e resultados numéricos relatados no Apêndice D [40] mostram a eficiência e efetividade da nova abordagem quando aplicada às duas versões do problema de escalonamento de tarefas divisíveis. Para ambos problemas foram comparados os resultados de oito configurações do BRKGA com reconexão por caminhos (BRKGA*) com o algoritmo BRKGA originalmente desenvolvido para cada problema.

Para o problema DLS-SR, forem selecionadas para os experimentos 60 dentre as maiores e mais difíceis instâncias, com 160 processadores escravos. Estas instâncias também foram usadas nos experimentos relatados em [8]. O BRKGA e todas as configurações do BRKGA* (com exceção da configuração 7) obtiveram soluções com o mesmo valor da melhor solução em todas as instancias testadas. Os resultados apresentados mostram que quando o BRKGA* consegue melhores tempos médios de execução do que o BRKGA original para encontrar a melhor solução, o primeiro é muito mais rápido. Porém, quando o BRKGA é mais rápido, a diferença para o tempo de execução de BRKGA* é pequena, em particular quando comparado com as configurações 3, 4 e 8 (que são aquelas que apresentam melhor desempenho para BRKGA*).

Para o problema DLS-MR, o BRKGA original foi comparado com as oito configurações de BRKGA* nas seis instancias apresentadas em [9]. Todos algoritmos foram executados dez vezes para cada instancia, com um tempo limite de 100 segundos para cada execução. Todas as configurações do BRKGA* (exceto a configuração 5 para algumas instancias) obtiveram soluções com valores médios melhores ou iguais aos obtidos pelo algoritmo BRKGA original.

Atualmente, como continuidade desta tese, as estratégias de aplicação de reconexão por caminhos no espaço das chaves aleatórias descritas neste capítulo estão sendo aplicadas aos problemas de otimização em quase-cliques. Resultados preliminares ratificam a eficiência da nova abordagem.

Capítulo 6

Conclusões

Nesta tese foram apresentados algoritmos (exatos e aproximados) para resolver dois problemas de otimização combinatória relacionados com quase-cliques: o problema de quaseclique de cardinalidade máxima (γ -MQCP) e o problema de particionamento mínimo de grafos em quase-cliques (γ -MQCPP).

O algoritmo exato apresentado para o γ -MQCP mostro ser competitivo sobre todo em instancias pouco densas quando comparado com os modelos e algoritmos existentes. Por outro lado, se obteve uma melhora do algoritmo do BRKGA-IG^{*} proposto anteriormente por Pinto, et al. [32] para o mesmo problema combinando o mesmo com uma busca local exata.

Demonstrou-se que o γ -MQCPP é NP-difícil, inclusive quando o número de quasecliques na partição é igual a dois. Para este problema se propõem quatro formulações MIP e uma heurística multi-partida gulosa randomizada com o objetivo de fornecer soluções iniciais viáveis para as formulações. O experimentos apresentados mostram que as formulações basadas no principio do representantes, obtiveram melhores soluções e provaram que a solução obtida é ótima em um maior numero de instancias. Também se mostra que instancias com valores de γ entre 0.7 e 0.4 são mais difíceis de serem resolvidas pelos modelos propostos.

Foi proposta a estrategia de reconexão de caminhos para BRKGAs no espaço das chaves aleatórias, que obteve bons resultados nos problemas de escalonamento de tarefas divisíveis. Como extensão futura de esta pesquisa, será desenvolvida uma heurística para o γ -MQCPP baseada no BRKGA com reconexão de caminhos no espaço das chaves aleatórias proposto nesta tese. Como resultado da tese, foram publicados dois artigos em periódicos relacionados ao problema γ -MQCP [31, 39], assim como submetido um artigo referente ao problema γ -MQCPP [25]. Além disso, foi publicado um artigo em congresso sobre o BRKGA com reconexão de caminhos [40].

Referências

- ABELLO, J.; PARDALOS, P. M.; RESENDE, M. G. C. On maximum clique problems in very large graphs. In <u>External Memory Algorithms</u> (1999), J. M. Abello and J. S. Vitter, Eds., American Mathematical Society, pp. 119–130.
- [2] ABELLO, J.; RESENDE, M.; SUDARSKY, S. Massive quasi-clique detection. In Proceedings of the 5th Latin American Symposium on the Theory of Informatics, J. Abello and J. Vitter, Eds., vol. 2286 of Lecture Notes in Computer Science. Springer, Berlin, 2002, pp. 598–612.
- [3] ANDRADE, C. E.; TOSO, R. F.; GONÇALVES, J. F.; RESENDE, M. G. The multi-parent biased random-key genetic algorithm with implicit path-relinking and its real-world applications. <u>European Journal of Operational Research 289</u> (2021), 17–30.
- [4] BASU, S.; SENGUPTA, D.; MAULIK, U.; BANDYOPADHYAY, S. A strong Nash stability based approach to minimum quasi clique partitioning. In <u>2014 Sixth International</u> <u>Conference on Communication Systems and Networks</u> (Bangalore, 2014), IEEE, pp. 1–6.
- [5] BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 2 (1994), 154–160.
- [6] BIGLIEL, T. Politnetz.ch, 2009. Online reference at https://de.wikipedia.org/wiki/Politnetz.ch.
- [7] BŁAŻEWICZ, J.; DROZDOWSKIAND, M.; MARKIEWICZ, M. Divisible task scheduling – Concept and verification. Parallel Computing 25 (1999), 87–98.
- [8] BRANDÃO, J. S.; NORONHA, T. F.; RESENDE, M. G. C.; RIBEIRO, C. C. A biased random-key genetic algorithm for single-round divisible load scheduling. <u>International</u> Transactions in Operational Research 22 (2015), 823–839.
- [9] BRANDÃO, J. S.; NORONHA, T. F.; RESENDE, M. G. C.; RIBEIRO, C. C. A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems. International Transactions in Operational Research 27 (2017), 1061–1077.
- [10] CHENG, Y. C.; ROBERTAZZI, T. G. Distributed computation with communication delay. IEEE Transactions on Aerospace and Electronic Systems 24 (1988), 700–712.
- [11] GARCIA, D.; ABISHEVA, A.; SCHWEIGHOFER, S.; SERDÜLT, U.; SCHWEITZER, F. Ideological and temporal components of network polarization in online political participatory media. <u>Policy & Internet 7</u> (2015), 46–79.

- [12] GLOVER, F. Tabu search and adaptive memory programming Advances, applications and challenges. In <u>Interfaces in Computer Science and Operations Research:</u> <u>Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies,</u> <u>R. S. Barr, R. V. Helgason, and J. L. Kennington, Eds. Springer, Boston, 1997,</u> pp. 1–75.
- [13] GLOVER, F.; LAGUNA, M.; MARTÍ, R. Fundamentals of scatter search and path relinking. Control and Cybernetics 39 (2000), 653–684.
- [14] GLOVER, F.; LAGUNA, M.; MARTI, R. Scatter search and path relinking: Advances and applications. In <u>Handbook of Metaheuristics</u>, F. Glover and G. A. Kochenberger, Eds. Springer, Boston, 2003, pp. 1–35.
- [15] GLOVER, F.; LAGUNA, M.; MARTÍ, R. Scatter search and path relinking: Foundations and advanced designs. In <u>New optimization techniques in engineering</u>, G. Onwubolu and B. Babu, Eds., vol. 141 of <u>Studies in Fuzzyness and Soft Computing</u>. Springer, Berlin, 2004, pp. 87–100.
- [16] GONCALVES, J. F.; RESENDE, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics 17 (2011), 487–525.
- [17] HU, H.; YAN, X.; HUANG, Y.; HAN, J.; ZHOU, X. J. Mining coherent dense subgraphs across massive biological networks for functional discovery. <u>Bioinformatics</u> 21 (2005), i213–i221.
- [18] KARP, R. M. Reducibility among combinatorial problems. In <u>Complexity of</u> <u>Computer Computations</u> (New York, 1972), R. E. Miller and J. W. Thatcher, Eds., <u>Plenum Press</u>, pp. 85–103.
- [19] LEE, C.-K.; HAMDI, M. Parallel image processing applications on a network of workstations. Parallel Computing 21 (1995), 137–160.
- [20] LEE, V. E.; RUAN, N.; JIN, R.; AGGARWAL, C. A survey of algorithms for dense subgraph discovery. In <u>Managing and mining graph data</u>. Springer, 2010, pp. 303– 336.
- [21] LI, P.; VEERAVALLI, B.; KASSIM, A. A. Design and implementation of parallel video encoding strategies using divisible load analysis. <u>IEEE Transactions on Circuits</u> and Systems for Video Technology 15 (2005), 1098–1112.
- [22] LIN, W.; LIANG, C.; WANG, J. Z.; BUYYA, R. Bandwidth-aware divisible task scheduling for cloud computing. <u>Software: Practice and Experience 44</u> (2014), 163– 174.
- [23] MARINELLI, F.; PIZZUTI, A.; ROSSI, F. LP-based dual bounds for the maximum quasi-clique problem. Discrete Applied Mathematics 296 (2021), 118–140.
- [24] MARQUÉS-SÁNCHEZ, P.; PINTO-CARRAL, A.; FERNÁNDEZ-VILLA, T.; VÁZQUEZ-CASARES, A.; LIÉBANA-PRESA, C.; BENÍTEZ-ANDRADES, J. A. Identification of cohesive subgroups in a university hall of residence during the covid-19 pandemic using a social network analysis approach. <u>Scientific Reports 11</u>, 1 (2021), 1–10. Referência online disponíivel em https://doi.org/10.1038/s41598-021-01390-4, última consulta em 17/02/2022.

- [25] MELO, R. A.; RIBEIRO, C. C.; RIVEAUX, J. The minimum quasi-clique partitioning problem: Complexity, formulations, and a greedy randomized heuristic, 2021. Submetido para publicação.
- [26] NORONHA, T. F.; RESENDE, M. G. C.; RIBEIRO, C. C. A biased random-key genetic algorithm for routing and wavelength assignment. <u>Journal of Global Optimization</u> 50 (2011), 503–518.
- [27] OLIVEIRA, A. B.; PLASTINO, A.; RIBEIRO, C. C. Construction heuristics for the maximum cardinality quasi-clique problem. In <u>Abstracts of the 10th Metaheuristics</u> International Conference (Singapore, 2013), p. 84.
- [28] PAJOUH, F. M.; MIAO, Z.; BALASUNDARAM, B. A branch-and-bound approach for maximum quasi-cliques. Annals of Operations Research 216 (2014), 145–161.
- [29] PATTILLO, J.; VEREMYEV, A.; BUTENKO, S.; BOGINSKI, V. On the maximum quasi-clique problem. Discrete Applied Mathematics 161 (2013), 244–257.
- [30] PATTILLO, J.; YOUSSEF, N.; BUTENKO, S. On clique relaxation models in network analysis. European Journal of Operational Research 226 (2013), 9–18.
- [31] PINTO, B. Q.; RIBEIRO, C. C.; RIVEAUX, J. A.; ROSSETI, I. A BRKGA-based matheuristic for the maximum quasi-clique problem with an exact local search strategy. RAIRO: Recherche Opérationnelle 55 (2021), S741 – S763.
- [32] PINTO, B. Q.; RIBEIRO, C. C.; ROSSETI, I.; PLASTINO, A. A biased randomkey genetic algorithm for the maximum quasi-clique problem. <u>European Journal of</u> Operational Research 271 (2018), 849–865.
- [33] RESENDE, M. G. C.; RIBEIRO, C. C. GRASP with path-relinking: Recent advances and applications. In <u>Metaheuristics: Progress as real problem solvers</u>, T. Ibaraki, K. Nonobe, and M. Yagiura, Eds. Springer, New York, 2005, pp. 29–63.
- [34] RESENDE, M. G. C.; RIBEIRO, C. C. Biased-random key genetic algorithms: An advanced tutorial. In <u>Proceedings of the 2016 Genetic and Evolutionary Computation</u> <u>Conference - GECCO'16 Companion Volume</u> (Denver, 2016), Association for Computing Machinery, pp. 483–514.
- [35] RESENDE, M. G. C.; RIBEIRO, C. C. Optimization by GRASP: Greedy Randomized Adaptive Search Procedures. Springer, New York, 2016.
- [36] RESENDE, M. G. C.; RIBEIRO, C. C.; GLOVER, F.; MARTÍ, R. Scatter search and path-relinking: Fundamentals, advances, and applications. In <u>Handbook of</u> <u>metaheuristics</u>, M. Gendreau and J.-Y. Potvin, Eds., 2nd ed. Springer, New York, 2010, pp. 87–107.
- [37] RIBEIRO, C. C.; MANIEZZO, V. Preface to the special issue on matheuristics: Model-based metaheuristics. <u>International Transactions in Operational Research 22</u> (2014), 1–1.
- [38] RIBEIRO, C. C.; RESENDE, M. G. C. Path-relinking intensification methods for stochastic local search algorithms. Journal of Heuristics 18 (2012), 193–214.

- [39] RIBEIRO, C. C.; RIVEAUX, J. An exact algorithm for the maximum quasi-clique problem. International Transactions in Operational Research 26 (2019), 2199–2229.
- [40] RIBEIRO, C. C.; RIVEAUX, J. A.; BRANDAO, J. S. Biased random-key genetic algorithms using path-relinking as a progressive crossover strategy. In <u>2021 5th</u> <u>International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence</u> (2021), Association for Computing Machinery, pp. 28–36.
- [41] RIBEIRO, C. C.; VIANNA, D. S. A hybrid genetic algorithm for the phylogeny problem using path-relinking as a progressive crossover strategy. <u>International</u> Transactions in Operational Research 16 (2009), 641–657.
- [42] SPEARS, W.; DE JONG, K. On the virtues of parameterized uniform crossover. In Proceedings of the Fourth International Conference on Genetic Algorithms (San Mateo, 1991), R. Belew and L. Booker, Eds., Morgan Kaufman, pp. 230–236.
- [43] SPIRIN, V.; MIRNY, L. A. Protein complexes and functional modules in molecular networks. Proceedings of the National Academy of Sciences 100 (2003), 12123–12128.
- [44] TOSO, R. F.; RESENDE, M. G. C. A C++ application programming interface for biased random-key genetic algorithms. <u>Optimization Methods and Software 30</u> (2015), 81–93.
- [45] TSOURAKAKIS, C.; BONCHI, F.; GIONIS, A.; GULLO, F.; TSIARLI, M. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In <u>Proceedings of the 19th ACM SIGKDD International Conference on Knowledge</u> Discovery and Data Mining (Chicago, 2013), pp. 104–112.
- [46] TURGAY, A.; YAKUP, P. Optimal scheduling algorithms for communication constrained parallel processing. vol. 2400 of <u>Lecture Notes in COmputer Science</u>. 2002, pp. 197–206.
- [47] VEREMYEV, A.; PROKOPYEV, O. A.; BUTENKO, S.; PASILIAO, E. L. Exact MIP-based approaches for finding maximum quasi-clique and dense subgraph. Computational Optimization and Applications 64 (2016), 177–214.
- [48] VERMA, A.; BUTENKO, S. Network clustering via clique relaxations: A community based approach. In <u>Graph Partitioning and Graph Clustering</u>, vol. 588 of Contemporary Mathematics. American Mathematical Society, 2013, pp. 129–139.
- [49] VERTELETSKYI, V.; YEN, T.-C.; IZMAYLOV, A. F. Measurement optimization in the variational quantum eigensolver using a minimum clique cover. <u>The Journal of</u> Chemical Physics 152 (2020), 124114.
- [50] WANG, R.; KRISHNAMURTHY, A.; MARTIN, R.; ANDERSON, T.; CULLER, D. Modeling communication pipeline latency. <u>ACM Sigmetrics Performance Evaluation</u> Review 26 (1998), 22–32.
- [51] YIN, C.; GU, H.; ZHANG, S. Measuring technological collaborations on carbon capture and storage based on patents: A social network analysis approach. <u>Journal</u> of Cleaner Production 274 (2020), 122867.

[52] ZHOU, Q.; BENLIC, U.; WU, Q. An opposition-based memetic algorithm for the maximum quasi-clique problem. <u>European Journal of Operational Research 286</u> (2020), 63–83.

APÊNDICE A – An exact algorithm for the maximum quasi-clique problem
Intl. Trans. in Op. Res. 00 (2019) 1–31 DOI: 10.1111/itor.12637 INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

O

An exact algorithm for the maximum quasi-clique problem

Celso C. Ribeiro and José A. Riveaux

Institute of Computing, Universidade Federal Fluminense, Niterói RJ 24210-346, Brazil E-mail: celso@ic.uff.br [Ribeiro]; jangel.riveaux@gmail.com [Riveaux]

Received 7 September 2018; received in revised form 21 January 2019; accepted 25 January 2019

Abstract

Given a graph G = (V, E) and a threshold $\gamma \in (0, 1]$, the maximum quasi-clique problem amounts to finding a maximum cardinality subset C^* of the vertices in V such that the edge density of the graph induced in G by C^* is greater than or equal to the threshold. This problem is NP-hard and has a number of applications in data mining, for example, in social networks or phone call graphs. In this work, we present an exact algorithm to solve this problem, based on a quasi-hereditary property. We also propose a new upper bound that is used for pruning the search tree. Numerical results show that the new approach is competitive and outperforms the best integer programming approaches in the literature. The new upper bound is consistently tighter than previously existing bounds.

Keywords: maximum cardinality quasi-clique problem; maximum quasi-clique problem; maximum γ -clique problem; maximum clique problem; graphs; graph density

1. Introduction

Let G = (V, E) be a graph defined by a vertex set V and an edge set $E \subseteq V \times V$. G is a complete graph if there is an edge in E connecting every two different vertices in V. The graph G(V') induced in G by $V' \subseteq V$ is that with vertex set V' and edge set formed by all edges of E with both ends in V'. For any $V' \subseteq V$, the subset $E(V') \subseteq E$ is formed by all edges of G with both ends in V'. Whenever the set V' is alternatively represented by a permutation Π of its vertices, we indistinctly refer to V' or Π . For example, we may indistinctly refer to E(V') or to $E(\Pi)$ as the edge set of the graph induced in G by V' or Π , respectively.

The density of graph G is given by $dens(G) = |E|/(|V| \times (|V| - 1)/2)$. For any vertex $v \in V$, we denote by $deg_G(v)$ the degree of vertex v, that is, the number of vertices adjacent to v in G. In addition, for any $v \in V$ and any $V' \subseteq V$, we denote by $deg_G(v, V')$ the number of vertices of V' that are adjacent to v in G.

A subset $C \subseteq V$ is a clique of G if the graph G(C) induced in G by C is complete. Given a graph G = (V, E), the *maximum clique problem* consists in finding a maximum cardinality clique of G. It was proved to be NP-hard by Karp (1972).

Published by John Wiley & Sons Ltd, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main St, Malden, MA02148, USA.

^{© 2019} The Authors.

International Transactions in Operational Research © 2019 International Federation of Operational Research Societies

Given a graph G = (V, E) and a threshold $\gamma \in (0, 1]$, a γ -clique (or quasi-clique) is any subset $C \subseteq V$ such that the density of G(C) is greater than or equal to γ . A γ -clique C is maximal if there is no other γ -clique C' such that C is strictly contained in C'. The maximum quasi-clique problem (MQCP) amounts to finding a maximum cardinality subset C^* of the vertices in V such that the density of the graph induced in G by C^* is greater than or equal to the threshold γ . This problem was proved to be NP-hard by Pattillo et al. (2013). The problem has many applications in data mining, for example, in social networks or phone call graphs (Abello et al., 1999, 2002).

In this work, we present an exact algorithm for solving the maximum quasi-clique problem, based on a quasi-hereditary property. Section 2 reviews exact formulations and algorithms for the problem. Section 3 presents a naïve algorithm for enumerating all feasible quasi-cliques of a graph without repetitions. Section 4 shows how to improve the enumerative algorithm by pruning some branches of the search tree and proposes a new upper bound. Section 5 presents the complete, exact *QClique* algorithm for the maximum quasi-clique problem. Computational experiments are reported in Section 6, in which the proposed approach is compared with existing exact methods. Concluding remarks are drawn in the last section. The new algorithm is competitive with the best formulations in Pattillo et al. (2013) and Veremyev et al. (2016) solved by CPLEX and with the branch-and-bound algorithm in Pajouh et al. (2014). The new upper bound is consistently tighter than previously existing bounds.

2. Related work and mathematical formulations

Pattillo et al. (2013) introduced some properties and upper bounds of γ -cliques. Property 1 below will be used in the design of the exact algorithm proposed in Section 3.

Definition 1. Consider a graph G = (V, E) that satisfies a property P. If there exists a vertex $v \in V$ such that the induced graph $G(V \setminus \{v\})$ also satisfies property P, then we say that P is a quasi-hereditary property and that property P displays quasi-heredity or quasi-inheritance.

Property 1. The graph property of having edge density greater than or equal to γ displays quasiinheritance, that is, any γ -clique with s > 1 vertices is a strict superset of a γ -clique with s - 1 vertices.

Figure 1 illustrates Property 1 on a graph with five nodes in Fig. 1(a), whose density is greater than or equal to 0.7. The removal of node 1 leads to the graph in Fig. 1(b), whose density is also greater than or equal to 0.7.

We summarize below the two best mixed integer programming formulations for the maximum γ -clique problem reported in Pattillo et al. (2013) and Veremyev et al. (2016). The first formulation is the one among those in Pattillo et al. (2013) that obtains the best results for dense graphs. The second formulation appeared in Veremyev et al. (2016) and presents the best results and the tighter relaxation for sparse graphs. It turned out to be the most consistent model in their experiments.

Model F1 (Pattillo et al., 2013) has a binary variable x_i associated to each vertex of the graph:

$$x_i = \begin{cases} 1, & \text{if vertex } v_i \in V \text{ belongs to the solution,} \\ 0, & \text{otherwise.} \end{cases}$$

© 2019 The Authors.



Fig. 1. Illustration of Property 1 on a graph G with five vertices. Both graphs have densities greater than or equal to 0.7.

It also considers a variable $y_{ij} = x_i \cdot x_j$ associated to each pair of vertices $i, j \in V$, with i < j, which is linearized as below:

Model F1:
$$\max_{i \in V} x_i$$
 (1)

subject to

$$\sum_{(i,j)\in E:i
⁽²⁾$$

$$y_{ij} \le x_i, \qquad \forall i, j \in V, \quad i < j, \tag{3}$$

$$y_{ij} \le x_j, \qquad \forall i, j \in V, \quad i < j, \tag{4}$$

$$y_{ij} \ge x_i + x_j - 1, \qquad i, j = 1, \dots, n, \quad i < j,$$
(5)

$$x_i \in \{0, 1\}, \qquad \forall i \in V, \tag{6}$$

$$y_{ii} \ge 0, \qquad \forall i, j \in V, \quad i < j. \tag{7}$$

The objective function (1) maximizes the number of vertices in the solution. If two vertices *i* and *j* belong to a solution, then $x_i = x_j = 1$ and $y_{ij} = x_i \cdot x_j = 1$. If edge $(i, j) \in E$, then it contributes to the density of the quasi-clique. Constraint (2) ensures that the density of the solution is greater than or equal to γ . Constraints (3) and (4) ensure that $y_{ij} = 0$ if any of the vertices *i* or *j* is not selected as a member of the quasi-clique. Constraints (5) ensure that any existing edge $(i, j) \in E$ will contribute to the solution if both of its extremities are chosen. Constraints (6) and (7) impose the integrality and nonnegativity requirements on the problem variables, respectively.

To introduce the second formulation, let ω^u and ω^ℓ be, respectively, any upper and lower bounds on the size of a maximum γ -clique in G. The lower bound can be set to 1 if there is no information available about the sizes of γ -cliques in G. Its value can be increased using the size of any heuristically identified quasi-clique, for example, it can be set to be the size of any clique in G. The upper bound

© 2019 The Authors.

can be simply set to the trivial value |V|, or we can use, for example, the result of Pattillo et al. (2013):

$$\omega^{\mu} = \left\lfloor \frac{1}{2} + \frac{1}{2} \sqrt{1 + 8 \frac{|E|}{\gamma}} \right\rfloor,\tag{8}$$

or, if the graph is connected,

$$\omega^{u} = \left\lfloor \frac{1}{2} + \frac{2 + \sqrt{(\gamma + 2)^{2} + 8\gamma(|E| - |V|)}}{2\gamma} \right\rfloor.$$
(9)

Model F3 (Veremyev et al., 2016) also makes use of the binary variables x_i . New variables y_{ij} , with i < j, are defined as

$$y_{ij} = \begin{cases} 1, & \text{if edge } (i, j) \in E, \text{ with } i < j, \text{ belongs to the quasi-clique,} \\ 0, & \text{otherwise.} \end{cases}$$

In addition, let z_k be a binary variable that determines the size of the current solution, namely, $z_k = 1$ if and only if the γ -clique has k vertices, 0 otherwise. Using this notation, the following formulation was proposed for finding a maximum γ -clique based on the classical value-disjunction idea (see, e.g., Nemhauser and Wolsey, 1988), which states that only one of a set of values can be taken by some variable. This is applied to the size of a maximum γ -clique:

Model F3:
$$\max_{i \in V} x_i$$
 (1 revisited)

subject to

$$\sum_{(i,j)\in E: i< j} y_{ij} \ge \gamma \cdot \sum_{k=\omega^l}^{\omega^u} \frac{k(k-1)}{2} z_k,$$
(10)

$$y_{ij} \le x_i, \quad \forall (i, j) \in E, \quad i < j,$$
 (3 revisited)

$$y_{ij} \le x_j, \quad \forall (i,j) \in E, \quad i < j,$$
 (4 revisited)

$$\sum_{i \in V} x_i = \sum_{k=\omega^l}^{\omega^*} k \cdot z_k,\tag{11}$$

$$\sum_{k=\omega^{l}}^{\omega^{*}} z_{k} = 1, \tag{12}$$

$$x_i \in \{0, 1\}, \quad \forall i \in V,$$
 (6 revisited)

$$y_{ij} \ge 0, \quad \forall i, j \in E, \quad i < j,$$
 (7 revisited)

$$z_k \ge 0, \quad \forall k \in \{\omega^l, \dots, \omega^u\}.$$
⁽¹³⁾

 $\ensuremath{\mathbb{O}}$ 2019 The Authors.

...11

Constraints (3) and (4) ensure that y_{ij} can be set to 1 only if both vertices *i* and *j* belong to the γ -clique, that is, $x_i = x_j = 1$. Constraint (10) represents the edge density requirements for the subgraph induced by the *k* chosen vertices, while constraints (11) and (12), together, enforce the appropriate value in the right-hand side of (10). As for model F1, variables y_{ij} can be relaxed to be continuous in constraint (7) due to the structure of constraints (10)-(3)-(4) and to the fact that the objective function maximizes a linear function with positive costs. Also, the binary restrictions for the z_k variables may be replaced by the nonnegativity constraints (13).

Pajouh et al. (2014) proposed an exact depth-first search-based enumeration algorithm for the maximum quasi-clique problem.

3. Naïve enumeration algorithm based on quasi-heredity

In this section, we introduce the fundamentals of an exact algorithm for the maximum quasi-clique problem based on Property 1, which states that if a γ -clique of size k does not exist in graph G, then no larger γ -clique exists as well.

Basically, this naïve algorithm enumerates all feasible quasi-cliques of the graph without repetitions, based on the following proposition:

Proposition 1. For any γ -clique C of cardinality k of graph G, there exists a permutation Π of the vertices in C such that, for every i = 1, ..., k, the last vertex v of the prefix Π_i of size i of Π has minimum degree in the induced subgraph $G(\Pi_i)$.

Proof. This result follows as a consequence of Property 1. It can be proved by a constructive algorithm that builds the permutation from scratch. First, select the vertex v' of C with minimum degree in G(C) to be the last vertex in Π and remove it from C. Then, the density of the new induced graph $G(C \setminus \{v'\})$ is greater than or equal to that of G(C) because the degree of the removed vertex does not increase the average degree of the graph (Pattillo et al., 2013). This step can be repeated iteratively and the algorithm finishes when all vertices have been selected and the resulting sequence meets the condition of Proposition 1.

Figure 2 illustrates Proposition 1 on a graph G with five vertices when a γ -clique for $\gamma = 0.7$ is sought. $C = V = \{1, 2, 3, 4, 5\}$ is a maximum 0.7-clique, since $dens(G(C)) = 7/10 \ge \gamma$. In Fig. 2(a), vertex 1 is selected as that with the smallest degree in G and placed in the last position of the permutation. The resulting subgraph has density $0.8\overline{3}$ and $\Pi = <1 >$. Next, vertex 2 is selected in Fig. 2(b) and the resulting permutation becomes $\Pi = <2, 1 >$. The remaining subgraph in Fig. 2(c) is a clique and any order of the vertices will meet the conditions of the proposition. In particular, the final permutation could be, for example, $\Pi = <3, 4, 5, 2, 1 >$. The subgraphs induced by the set of vertices of permutations $\Pi_1 = <3 >$, $\Pi_2 = <3, 4 >$, and $\Pi_3 = <3, 4, 5 >$ have their densities equal to 1, the subgraph induced by the vertices of $\Pi_4 = <3, 4, 5, 2, 1 >$ has its density equal to $0.8\overline{3}$, and that induced by the vertices of $\Pi_5 = \Pi = <3, 4, 5, 2, 1 >$ has density 0.7.

The enumeration procedure amounts to constructing the optimal vertex permutation sequence that leads to the maximum γ -clique. The last vertex in every prefix of the permutation is always that with minimum degree, but in case there are ties and two or more vertices have minimum degree, the



Fig. 2. Illustration of Proposition 1 on a graph G with five vertices when a maximum γ -clique for $\gamma = 0.7$ is sought. $C = V = \{1, 2, 3, 4, 5\}$ is a maximum 0.7-clique, since $dens(G(C)) = 7/10 \ge \gamma$.

optimal sequence might not be unique. To avoid this situation, we suppose that the vertices of V are labeled with 1, 2, ..., |V| and, in case of ties, the enumeration always selects that with the largest label. Therefore, the sequence that represents a feasible solution will be unique.

Algorithm 1 below is a backtracking strategy that enumerates all γ -cliques in G without repetition. The first call to the algorithm is performed with an empty permutation $\Pi = <>$ and an empty γ clique $C^* = \emptyset$. In line 1, function *getCandidates* generates the candidate list *CL* formed by all vertices that can be inserted at the end of the current permutation Π . The candidate list returned after the first call to *getCandidates* is CL = V. The recursion is interrupted in line 3 when the candidate list becomes empty and the best γ -clique is the current C^* . The loop in lines 5–11 investigates the extension of the current permutation Π by each candidate $j \in CL$. A new permutation Π' is obtained by appending vertex j as the last one of the current permutation Π . Line 7 compares the size of the new permutation Π' with that of the best known solution C^* . If the new solution improves upon the current best, then the vertices in Π' will form the new best γ -clique C^* in line 8. A recursive call is made in line 10.

A 1	•	41	- 1	D		1.
- A	anri	thm		Rad	ztra	alzina
	IYUI I				<i>\</i> ((n_{111})
	_			2000		0.00.00

Req	uire: G, Π, γ, C^*
1:	$CL \leftarrow getCandidates(G, \Pi, \gamma)$
2:	if $CL = \emptyset$ then
3:	return
4:	end if
5:	for all $j \in CL$ do
6:	$\Pi' \leftarrow \Pi \oplus j$
7:	if $ \Pi' > C^* $ then
8:	$C^* \leftarrow \Pi'$
9:	end if
10:	Backtracking(G, Π', γ, C^*)
11:	end for

© 2019 The Authors.

Function *getCandidates* called by the above algorithm builds the candidate list *CL* formed by the vertices that can be inserted at the end of a prefix. Algorithm 2 presents its pseudo-code in detail. Line 1 initializes the candidate list *CL* as empty and line 2 initializes *i* as the size of the current permutation Π . The loop in lines 3–10 considers the addition to the current permutation Π of all vertices in $V \setminus \Pi$. We recall that $deg_G(j, \Pi)$ denotes the number of vertices of the current permutation Π that are adjacent to vertex *j* in *G* and $E(\Pi)$ denotes the edge set of $G(\Pi)$. For any candidate vertex *j*, line 4 checks if the subgraph $G(\Pi \oplus j)$ induced by the vertices of Π and *j* is a γ -clique. Line 5 creates a tentative sequence Π' by appending vertex *j* at the end of sequence Π . Line 6 determines if there is another vertex $w \in \Pi$ with degree smaller than *j* or with the same degree as *j* but with a larger label. If this is not the case, then vertex *j* is definitely added to the candidate list *CL* in line 7. The candidate list *CL* is returned in line 11.

Algorithm 2. getCandidates

Require: G, Π, γ 1: $CL \leftarrow \emptyset$ 2: $i \leftarrow |\Pi|$ 3: for all $j \in V \setminus \Pi$ do 4: if $deg_G(j, \Pi) + |E(\Pi)| \ge \gamma (i+1)i/2$ then 5: $\Pi' \leftarrow \Pi \oplus j$ 6: if $\{w \in \Pi : deg_G(w, \Pi') < deg_G(j, \Pi') \text{ or } [deg_G(w, \Pi') = deg_G(j, \Pi') \text{ and } w > j]\} = \emptyset$ then 7: $CL \leftarrow CL \cup \{j\}$ 8: end if 9: end if 10: end for 11: return CL

Algorithms 1 and 2 produce an enumeration tree. A path from the root to any other node of this tree corresponds to a permutation Π that represents a solution to MQCP. Every feasible solution *C* is represented by exactly one path in the tree. Figure 3 illustrates the tree produced for the graph in Fig. 2, with $\gamma = 0.8$.

4. Pruning and a new upper bound for the maximum quasi-clique problem

In this section, we show how the enumeration algorithm can be accelerated by pruning of the search tree. In the following, we assume that we are currently investigating all feasible solutions associated with vertex permutations that have a common prefix Π_i of size i < k = LB + 1, where LB is a known lower bound to the optimal solution value. If all such permutations (i.e., those with a common prefix Π_i of size i) have fewer than k vertices, then the node corresponding to this permutation can be pruned because no solution obtained by branching from it can improve the current lower bound.

The remainder of this section is organized as follows. Section 4.1 will introduce the underlying principle of the pruning strategy, which is based on the computation of an upper bound to the number of edges of a solution of size k represented by a permutation with prefix Π_i . Next, Section 4.2



Fig. 3. Enumeration tree produced by Algorithms 1 and 2 for graph G of Fig. 2, with $\gamma = 0.8$.

will present the computation of an upper bound to the number of edges between the vertices in the prefix Π_i and those outside the prefix in a solution of size k. Finally, Section 4.3 introduces the pruning criterion for the naïve enumeration algorithm originally proposed in Section 3, leading to a stronger upper bound for the maximum quasi-clique problem.

4.1. Upper bound to the number of edges in a γ -clique with prefix Π_i

The pruning strategy assumes the existence of a permutation Π_k with prefix Π_i , corresponding to a γ -clique with size k > i.

Let us suppose the existence of an upper bound to the number of edges in $G(\Pi_k)$. If this upper bound is not enough to define a γ -clique, then such a permutation Π_k does not exist.

Let $E(\Pi_i, \Pi_k \setminus \Pi_i)$ denote the set of edges in G between the vertices of Π_i and $\Pi_k \setminus \Pi_i$. If Π_k corresponds to a γ -clique, then the following condition holds and gives a lower bound to $|E(\Pi_k)|$:

$$|E(\Pi_k)| = |E(\Pi_i) \cup E(\Pi_i, \Pi_k \setminus \Pi_i) \cup E(\Pi_k \setminus \Pi_i)| \ge \gamma \cdot \binom{k}{2}$$

Then,

$$|E(\Pi_k)| = |E(\Pi_i)| + |E(\Pi_i, \Pi_k \setminus \Pi_i)| + |E(\Pi_k \setminus \Pi_i)|.$$
(14)

Since $|E(\Pi_i)|$ is known, in order to calculate an upper bound to $|E(\Pi_k)|$ we need to calculate an upper bound to $|E(\Pi_i, \Pi_k \setminus \Pi_i)| + |E(\Pi_k \setminus \Pi_i)|$. An upper bound to the first term $|E(\Pi_i, \Pi_k \setminus \Pi_i)|$ will be derived in Section 4.2. Finally, Section 4.3 will complete the development of the upper bound.

© 2019 The Authors.



Fig. 4. Permutation Π_i is a prefix of permutation Π_k : vertex v_i has minimum degree in $G(\Pi_i)$.

4.2. Upper bound to the number of edges between Π_i and $\Pi_k \setminus \Pi_i$

The development of an upper bound to the number of edges between Π_i and $\Pi_k \setminus \Pi_i$ can be split into two main steps. In the first step, we give a straightforward upper bound to the number of edges between Π_i and $\Pi_k \setminus \Pi_i$ in Section 4.2.1. We also indicate how this bound can be improved by using an alternative sequence of vertices Π' such that the number of edges between Π_i and Π' gives a tighter upper bound to the number of edges between Π_i and $\Pi_k \setminus \Pi_i$. In the second step, we propose in Section 4.2.2 a greedy algorithm to compute a sequence Π' leading to this improved bound.

4.2.1. First upper bound

As in Section 4.1, we assume the existence of a permutation Π_k as illustrated in Fig. 4. Permutation Π_i is a prefix of permutation Π_k , with i < k = LB + 1. The last vertex of every prefix of Π_k has minimum degree in the graph induced in *G* by the nodes in this prefix. Therefore, vertex v_{i+j} has minimum degree in the graph induced by $\Pi_{i+j} = \langle v_1, \ldots, v_i, v_{i+1}, \ldots, v_{i+j} \rangle$. Let *d* be the degree of the last vertex v_i of Π_i in $G(\Pi_i)$.

For every j = 1, ..., k - i, $deg_G(v_{i+j}, \Pi_i)$ is bounded by $\min\{i, d+j\}$: it can neither be greater than the number *i* of vertices in Π_i nor larger than d + j because in this case $deg_G(v_i, \Pi_{i+j})$ would be smaller than $deg_G(v_{i+j}, \Pi_{i+j})$. Therefore, $\sum_{j=1}^{k-i} \min\{i, d+j\}$ gives a first upper bound to the number of edges between Π_i and $\Pi_k \setminus \Pi_i$.

This bound can be further improved. We denote by $V' = V \setminus \prod_i$ the subset of vertices of the graph that do not appear in permutation \prod_i . In addition, we denote by $\prod' = \langle f_1, \ldots, f_{k-i} \rangle$ any sequence of k - i vertices in V' such that for every $j = 1, \ldots, k - i$, the number of vertices of \prod_i that are adjacent to f_1, \ldots, f_j in G is greater than or equal to the number of vertices of \prod_i that are adjacent to v_{i+1}, \ldots, v_{i+j} in G, as illustrated in Fig. 5.

In the following, we propose a greedy algorithm to build the sequence Π' .

4.2.2. *Greedy algorithm for building sequence* Π'

We use the following strategy to build sequence Π' : select from V' the yet unselected vertex f_j whose number of adjacent vertices of Π_i in G is maximum and not greater than d + j at each iteration j = 1, ..., k - i. Then, Proposition 2 holds:

Proposition 2. Recall that $\Pi' = \langle f_1, ..., f_{k-i} \rangle$. Then, $\sum_{p=1}^{j} \deg_G(v_{i+p}, \Pi_i) \leq \sum_{p=1}^{j} \deg_G(f_p, \Pi_i)$, with j = 1, ..., k - i.

© 2019 The Authors.



Fig. 5. The number of vertices in Π_i adjacent to the vertices f_1, \ldots, f_j in *G* is greater than or equal to the number of vertices in Π_i adjacent to the vertices v_{i+1}, \ldots, v_{i+j} in *G*, for $j = 1, \ldots, k - i$.

Proof. We show in the following that there exists an exact matching between the vertices $\{v_{i+1}, \ldots, v_{i+j}\}$ of Π_k and the vertices $\{f_1, \ldots, f_j\}$ of Π' in the sequence constructed by the greedy algorithm, in which the number of vertices adjacent to Π_i in G of each vertex of the first set is smaller than or equal to that of the corresponding vertex in the second.

If $deg_G(f_j, \Pi_i) \ge deg_G(v_{i+j}, \Pi_i)$ for every j = 1, ..., k - i, then it is possible to match every vertex v_{i+j} with f_j .

Otherwise, let f_j be a vertex of Π' such that $deg_G(f_j, \Pi_i) < deg_G(v_{i+j}, \Pi_i)$. Suppose vertex f_j was selected by the greedy algorithm as the next unselected vertex of V' with the maximum number of adjacent vertices in Π_i that is not greater than d + j. If v_{i+j} was still unselected, then it would have been selected as f_j . Therefore, v_{i+j} was already selected in a previous iteration (i.e., there exists $1 \le j' < j : f_{j'} = v_{i+j}$) and there is at least one vertex from $\{v_{i+1}, \ldots, v_{i+j-1}\}$ that remains unselected in V'.

If $v_{i+j'} \notin \{f_1, \ldots, f_j\}$, then it remains unselected in V' and its degree is smaller than d + j. Therefore, $deg_G(f_j, \Pi_i) \ge deg_G(v_{i+j'}, \Pi_i)$ and it is possible to match $v_{i+j'}$ with f_j and v_{i+j} with $f_{j'}$. Contrarily, if $v_{i+j'} \in \{f_1, \ldots, f_j\}$, the matching procedure continues until it finds a yet unselected vertex v in $\{v_{i+1}, \ldots, v_{i+j}\}$. Since $d + j \ge deg_G(v, \Pi_i)$ and $deg_G(f_j, \Pi_i) \ge deg_G(v, \Pi_i)$, vertices v and f_j can be matched, as shown in Fig. 6.

The process is applied to all unmatched vertices $f_j \in \Pi'$ with $deg_G(f_j, \Pi_i) < deg_G(v_{i+j}, \Pi_i)$. Finally, all unmatched vertices $f_j \in \Pi'$ such that $deg_G(f_j, \Pi_i) \ge deg_G(v_{i+j}, \Pi_i)$ are matched with vertex v_{i+j} .

Consequently,
$$\sum_{p=1}^{j} \deg_{G}(f_{p}, \Pi_{i}) \ge \sum_{p=1}^{j} \deg_{G}(v_{i+p}, \Pi_{i}).$$

Proposition 3 also holds if this strategy is used to build sequence Π' :

Proposition 3. Recall that $\Pi' = \langle f_1, ..., f_{k-i} \rangle$. If $deg_G(f_j, \Pi_i) \neq \min\{i, d+j\}$, then $deg_G(f_j, \Pi_i) \leq deg_G(f_{j-1}, \Pi_i)$, for j = 1, ..., k - i.

© 2019 The Authors.



Fig. 6. Existence of an exact matching between the vertices v_{i+1}, \ldots, v_{i+j} of Π_k and the vertices f_1, \ldots, f_j of Π' .



Fig. 7. Characterization of vertex w in sequence Π' , showing that f_j should have been selected instead of w as the hth vertex in Π' .

Proof. We assume that f_j is such that $deg_G(f_{j-1}, \Pi_i) < deg_G(f_j, \Pi_i) < \min\{i, d+j\}$ for some $j = 1, \ldots, k-i$.

In case $deg_G(f_j, \Pi_i) \leq deg_G(v_i, \Pi_i)$, let w be the first vertex in sequence f_1, \ldots, f_{j-1} such that $deg_G(w, \Pi_i) < deg_G(f_j, \Pi_i)$. Otherwise, $deg_G(f_j, \Pi_i) = deg_G(v_i, \Pi_i) + p$, where 0 . In this case, we define <math>w as the first vertex in the sequence f_p, \ldots, f_{j-1} such that $deg_G(w, \Pi_i) < deg_G(f_j, \Pi_i)$. In both cases, there exists at least one vertex meeting this condition, because we assumed that $deg_G(f_{j-1}, \Pi) < deg_G(f_j, \Pi_i)$. If h is the position of w in Π' , as shown in Fig. 7, then $deg_G(f_j, \Pi_i) < d + h$. Therefore, there is a contradiction because vertex f_j would have been selected (instead of vertex w) as that with the maximum number of adjacent vertices of Π_i in G by the greedy algorithm.

The greedy algorithm used in order to compute the sequence Π' is based on the counting sort (see, e.g., Cormen et al., 2001). Initially, the algorithm partitions $V' = V \setminus \Pi_i$ into i + 1 different sets labeled B_0 to B_i . Each set B_j contains the vertices in V' with j adjacent vertices of G in Π_i , for j = 0, ..., i. Each step j = 1, ..., k - i of the algorithm starts from set B_ℓ . For the first step, $B_\ell = B_{d+1}$, with $\ell = d + 1$, where $d = deg_G(v_i, \Pi_i)$ as it can be seen in Fig. 8(a). The algorithm moves from right to left, examining sets B_ℓ , $B_{\ell-1}, ..., B_0$ until it finds the first

© 2019 The Authors.

International Transactions in Operational Research © 2019 International Federation of Operational Research Societies





(c) If set B_{d+j} is empty, then algorithm does not move from the current set

Fig. 8. Illustration of counting sort-based algorithm for building Π' .

nonempty set B_{ℓ}^* , from where it randomly selects and removes a vertex that will be the next vertex of Π' . Next, the algorithm resets $\ell = \min\{i, d+j\}$ if $B_{\min\{i, d+j\}} \neq \emptyset$ (Fig. 8(b)); $\ell = \ell^*$ otherwise (Fig. 8(c)). Then, a new step resumes. If the algorithm reaches set B_0 at any time with $B_0 = \emptyset$, then Π' cannot be constructed and a solution Π_k of size k > i containing Π_i as a prefix does not exist.

The pseudo-code in Algorithm 3 shows how Π' is built following the counting sort strategy. In line 1, the sequence Π' is initialized. Sets B_0, \ldots, B_i are initialized as empty in line 2. The degree of the last vertex v_i in Π_i is calculated in line 3. The loop in lines 4–7 creates the sets containing all vertices of *G* with the same number of adjacent vertices in Π_i . Line 8 defines the set B_ℓ from where the search starts. The loop in lines 9–22 performs as many iterations as needed to build sequence Π' . The loop in lines 10 and 11 detects if there is a nonempty set B_ℓ from where a vertex will be selected. If this is not the case, an empty sequence is returned indicating that a sequence Π' does not exist. Otherwise, a vertex v is randomly selected and removed from B_ℓ in lines 15 and 16, respectively. Next, vertex v is added to the end of sequence Π' in line 17. If $B_{\min\{i,d+j\}} \neq \emptyset$ the algorithm resets $\ell = \min\{i, d + j\}$ in line 19, before resuming a new iteration of the loop from set B_ℓ . Finally, line 23 returns the sequence Π' that gives the tighter upper bound to the number of edges between Π_i and $\Pi_k \setminus \Pi_i$.

© 2019 The Authors.

Algorithm 3. buildMaxAdjacentSequence

Require: G, Π_i, v_i, k 1: Π′ ←<> 2: Create sets B_0, \ldots, B_i 3: $d \leftarrow deg_{G(\Pi_i)}(v_i)$ 4: for all $v \in V \setminus \Pi_i$ do 5: $box \leftarrow deg_G(v, \Pi_i)$ 6: $B_{box} \leftarrow B_{box} \cup \{v\}$ 7: end for 8: $\ell \leftarrow d + 1$ 9: **for** j = 1 to k - i **do** 10: while $B_{\ell} = \emptyset$ and $\ell > 0$ do $\ell \leftarrow \ell - 1$ 11: end while $\mathbf{if}B_\ell = \emptyset$ then 12: 13: return <> 14: else 15: Randomly select $v \in B_{\ell}$ $\begin{array}{l} B_\ell \leftarrow B_\ell \setminus \{v\} \\ \Pi' \leftarrow \Pi' \oplus v \end{array}$ 16: 17: 18: if $B_{\min\{i,d+j\}} \neq \emptyset$ then 19: $\ell \leftarrow \min\{i, d+j\}$ 20: end if 21: end if 22: end for 23: return П'

Sets B_0, \ldots, B_i are created in time O(n). Since the algorithm visits at most *i* sets in each step $j = 1, \ldots, k - i$, the construction of Π' takes time $O(i \cdot (k - i))$. Then, Algorithm 3 runs in time $O(n + i \cdot (k - i))$.

Finally, Algorithm 3 builds a sequence Π' used to calculate an upper-bound to $|E(\Pi_i, \Pi_k \setminus \Pi_i)|$. By definition,

$$|E(\Pi_i, \Pi_k \setminus \Pi_i)| = \sum_{p=1}^{k-i} deg_G(v_{i+p}, \Pi_i).$$

Then, from Proposition 2, sequence $\Pi' = \langle f_1, \ldots, f_{k-i} \rangle$ satisfies

$$\sum_{p=1}^{j} deg_{G}(v_{i+p}, \Pi_{i}) \le \sum_{p=1}^{j} deg_{G}(f_{p}, \Pi_{i}), \quad j = 1, \dots, k-i.$$
(15)

Finally, taking j = k - i in inequality (15) gives an upper bound to $|E(\Pi_i, \Pi_k \setminus \Pi_i)|$:

$$|E(\Pi_i, \Pi_k \setminus \Pi_i)| \le \sum_{p=1}^{k-i} deg_G(f_p, \Pi_i).$$

© 2019 The Authors.

4.3. Calculating an upper bound to the number of edges in $G(\Pi_k)$

We assumed in Section 4.1 that $G(\Pi_k)$ is a γ -clique associated with a permutation $\Pi_k = <$ $v_1, \ldots, v_i, v_{i+1}, \ldots, v_k >$. We are searching for an upper bound to the number of edges in $G(\Pi_k)$. Although inequality (15) gives an upper bound to $|E(\Pi_i, \Pi_k \setminus \Pi_i)|$, we still need an upper bound to $|E(\Pi_i, \Pi_k \setminus \Pi_i)| + |E(\Pi_k \setminus \Pi_i)|$. We recall that $d = deg_G(v_i, \Pi_i)$.

Let $\Pi'' = \langle f'_1, \dots, f'_{k-i} \rangle$ be a sequence of vertices of $V' = V \setminus \Pi_i$, such that for any $j = 1, \dots, k-i, f'_j$ is a vertex from V' with maximum degree in G and $deg_G(f'_j, \Pi_i) \leq d+j$. Following the same argument of the proof of Proposition 2, there exists an exact matching between the vertices $\{v_{i+1}, \ldots, v_{i+j}\}$ of Π_k and the vertices $\{f'_1, \ldots, f'_j\}$ of Π'' , in which

$$\sum_{p=1}^{j} deg_G(v_{i+p}) \le \sum_{p=1}^{j} deg_G(f_p'), \quad j = 1, \dots, k-i.$$
(16)

Sequence $\Pi'' = \langle f'_1, \ldots, f'_{k-i} \rangle$ can be computed by Algorithm 4. Line 1 initializes V' as $V \setminus \Pi_i$. Line 2 sets d as the degree of the last vertex v_i in Π_i . Line 3 initializes Π'' as an empty sequence. The loop in lines 5–9 builds the sequence Π'' . At each iteration j = 1, ..., k - i, a new vertex $f'_i \in V'$ is selected in line 5, such that its degree is maximum in G and $deg_G(f'_i, \Pi_i) \le d + j$. Vertex f'_i is inserted into Π'' in line 6 and removed from V' in line 7. Finally, the algorithm returns sequence Π'' in line 9.

Algorithm 4. buildMaxDegreeSequence

Require: G, Π_i, v_i, k 1: $V' \leftarrow V \setminus \Pi_i$ 2: $d \leftarrow deg_G(v_i, \Pi_i)$ 3: Π″ ←<> 4: for j = 1, ..., k - i do 5: Select $f'_i \in V$ with maximum degree in G such that $deg_G(f'_i, \Pi_i) \leq d + j$ $\Pi'' \leftarrow \Pi'' \oplus f'_i$ 6: $V' \leftarrow V' \setminus \{f'_i\}$ 7: 8: end for 9: return Π["]

Since Π_i is a prefix of $\Pi_{i+j} = \langle v_1, \ldots, v_i, v_{i+1}, \ldots, v_{i+j} \rangle$ and $\Pi_{i+j} \subseteq V$,

$$deg_G(v_{i+p}, \Pi_i) + deg_G(v_{i+p}, \Pi_{i+j} \setminus \Pi_i) \le deg_G(v_{i+p}), \quad p = 1, \dots, j.$$

Adding the above inequalities for p = 1, ..., j, we obtain

$$\sum_{p=1}^{j} deg_{G}(v_{i+p}, \Pi_{i}) + \sum_{p=1}^{j} deg_{G}(v_{i+p}, \Pi_{i+j} \setminus \Pi_{i}) \le \sum_{p=1}^{j} deg_{G}(v_{i+p}).$$
(17)

© 2019 The Authors.

By transitivity, from inequalities (16) and (17):

$$\sum_{p=1}^{j} \deg_{G}(v_{i+p}, \Pi_{i}) + \sum_{p=1}^{j} \deg_{G}(v_{i+p}, \Pi_{i+j} \setminus \Pi_{i}) \le \sum_{p=1}^{j} \deg_{G}(f_{p}').$$
(18)

Adding inequalities (15) and (18), we obtain a new inequality after some manipulations:

$$\sum_{p=1}^{j} deg_{G}(v_{i+p}, \Pi_{i}) + \sum_{p=1}^{j} \frac{deg_{G}(v_{i+p}, \Pi_{i+j} \setminus \Pi_{i})}{2}$$

$$\leq \sum_{p=1}^{j} deg_{G}(f_{p}, \Pi_{i}) + \frac{\sum_{p=1}^{j} deg_{G}(f_{p}') - \sum_{p=1}^{j} deg_{G}(f_{p}, \Pi_{i})}{2}.$$
(19)

Since the sum of the degrees of the vertices of any graph is bounded by twice the number of its edges,

$$|E(\Pi_{i+j} \setminus \Pi_i)| = \sum_{p=1}^{j} \frac{deg_G(v_{i+p}, \Pi_{i+j} \setminus \Pi_i)}{2} \le \binom{j}{2}, \quad j = 1, \dots, k-i.$$
(20)

Hence, combining inequalities (15), (19), and (20):

$$\begin{split} &\sum_{p=1}^{j} deg_{G}(v_{i+p}, \Pi_{i}) + \sum_{p=1}^{j} \frac{deg_{G}(v_{i+p}, \Pi_{i+j} \setminus \Pi_{i})}{2} \\ &\leq \sum_{p=1}^{j} deg_{G}(f_{p}, \Pi_{i}) + \min\left\{\frac{\sum_{p=1}^{j} deg_{G}(f_{p}') - \sum_{p=1}^{j} deg_{G}(f_{p}, \Pi_{i})}{2}, \binom{j}{2}\right\}. \end{split}$$

Finally, taking j = k - i in the above inequality gives the upper bound:

$$|E(\Pi_{i}, \Pi_{k} \setminus \Pi_{i})| + |E(\Pi_{k} \setminus \Pi_{i})| \leq \sum_{p=1}^{k-i} deg_{G}(f_{p}) - \sum_{p=1}^{k-i} deg_{G}(f_{p}, \Pi_{i}), {k-i \choose 2}.$$
(21)

We recall that the number of edges in a solution of size k with prefix Π_i is given by

$$|E(\Pi_k)| = |E(\Pi_i)| + |E(\Pi_i, \Pi_k \setminus \Pi_i)| + |E(\Pi_k \setminus \Pi_i)|.$$
(14 revisited)

The first term in the right-hand side of this equation is the number of edges in $G(\Pi_i)$, while an upper bound to the sum of the second and third terms is given by expression (21). Therefore, if the inequality below is not satisfied for at least one value of j = 1, ..., k - i, then a solution of size k

 $\ensuremath{\mathbb{C}}$ 2019 The Authors.

15

International Transactions in Operational Research © 2019 International Federation of Operational Research Societies

with prefix Π_i does not exist and we can prune the current node of the search tree:

$$|E(\Pi_i)| + \sum_{p=1}^j deg_G(f_p, \Pi_i) + \min\left\{\frac{\sum_{p=1}^j deg_G(f_p') - \sum_{p=1}^j deg_G(f_p, \Pi_i)}{2}, \binom{j}{2}\right\}$$

$$\geq \gamma \cdot \binom{i+j}{2}.$$
(22)

Inequality (22) leads to a new upper bound *newUB* to the maximum quasi-clique problem. If $\Pi_0 = <>$ denotes an empty solution, then $|E(\Pi_0)| = 0$, $deg_G(v, \Pi_0) = 0$ for all $v \in V' = V \setminus \Pi_0$, and the sequence Π'' will contain the k vertices with the largest degrees in G. The upper bound *newUB* is given by the largest value of k = 1, ..., |V| such that $\sum_{v \in \Pi''} deg_G(v)/2 \ge \gamma \cdot {k \choose 2}$, that is,

$$newUB = \max\{k : k = 1, ..., |V| \text{ and } \sum_{v \in \Pi''} deg_G(v)/2 \ge \gamma \cdot k(k-1)/2\}.$$
 (23)

5. Exact algorithm *QClique*

The pseudo-code of the exact algorithm for the maximum quasi-clique problem is presented in Algorithm 5.

The initial parameters are the graph G = (V, E), the threshold γ , the current permutation Π_i that is initially an empty set, a lower bound LB, and the incumbent (best known solution) C^* , which is also initially an empty set. Line 1 sets i to the size of the current permutation Π_i . Line 2 checks if the current solution is not empty. In this case, lines 3-11 apply the pruning strategy to the current node of the search tree. Line 3 sets k to LB + 1. Line 4 calls algorithm buildMaxAd*jacentSequence* to compute a sequence Π' . If this algorithm returns an empty sequence, then the current node is pruned in line 5 because there are not enough vertices in $V' = V \setminus \Pi$ to build a feasible solution of size k. Line 7 calls algorithm *buildMaxDegreeSequence* to compute a sequence Π'' . Both sequences Π' and Π'' will be used in the loop in lines 8–11 to eventually prune the current node of the search three if inequality (22) is not satisfied for at least one value of j = 1, ..., k - i, because in that case a feasible solution with size greater than the current lower bound LB cannot be obtained by extending the current permutation Π_i . Otherwise, in case the node is not pruned, algorithm getCandidates generates in line 13 the candidate list CL with the vertices that can be used to extend the current solution. Line 14 checks if the candidate list is empty, in which case the current solution cannot be extended. Otherwise, line 17 checks if the extension of the current solution by any of the members of the candidate list improves the current lower bound. If this is the case, the lower bound LB is updated in line 18, a vertex $v \in CL$ is randomly selected in line 19, and the incumbent C^* is updated in line 20. The loop in lines 22–25 creates a new permutation Π_{i+1} for each candidate $j \in CL$ in line 23 and makes a recursive call to algorithm *QClique* in line 24. When the recursion terminates, C^* contains the best γ -clique obtained by the algorithm.

© 2019 The Authors.

International Transactions in Operational Research © 2019 International Federation of Operational Research Societies

16

Algorithm 5. QClique

Require: $G, \gamma, \Pi_i, LB, C^*$ 1: $i \leftarrow |\Pi_i|$ 2: **if** i > 0 **then** 3: $k \leftarrow LB + 1.$ 4: $\Pi' \leftarrow buildMaxAdjacentSequence(G, \Pi_i, v_i, k)$ 5: if $\Pi' = <>$ then return 6: end if $\Pi'' \leftarrow buildMaxDegreeSequence(G, \Pi_i, v_i, k)$ 7: for j = 1, ..., k - i do 8: 9: if inequality (22) is violated then return 10: end if end for 11: 12: end if 13: $CL \leftarrow getCandidates(G, \Pi_i, \gamma)$ 14: if $CL = \emptyset$ then 15: return 16: else 17: **if** $|\Pi_i| + 1 > LB$ then $LB \leftarrow |\Pi_i| + 1$ 18: 19: Randomly select $v \in CL$ 20: $C^* \leftarrow \Pi_i \oplus v$ 21: end if 22: for all $w \in CL$ do 23: $\Pi_{i+1} \leftarrow \Pi_i \oplus w$ 24: $QClique(G, \gamma, \Pi_{i+1}, LB, C^*)$ 25: end for 26: end if

Example. We present next an example of the application of algorithm *QClique* for solving the maximum quasi-clique problem on graph G with 8 vertices and 14 edges depicted in Fig. 9 for $\gamma = 0.8$. Figure 10 displays the search tree produced by algorithm *QClique*. Outer labels indicate the order in which the search nodes are investigated and explored by algorithm *QClique*, while inner labels represent the vertices considered to be added to the solution at each step. The optimal solution is the quasi-clique $C^* = \{1, 4, 6, 7, 3\}$ with density equal to 0.8. Only the 17 first search tree nodes are represented. Dashed edges follow pruned nodes. All other nonrepresented nodes will also be pruned.



Fig. 9. Graph G with 8 vertices and 14 edges.



Fig. 10. Partial search tree produced by the application of algorithm *QClique* to graph *G* with 8 vertices and 14 edges for $\gamma = 0.8$. Outer labels indicate the order in which the vertices are investigated and explored by algorithm *QClique*: the optimal solution is formed by the quasi-clique $C^* = \{1, 4, 6, 7, 3\}$ with density equal to 0.8. Dashed edges follow pruned nodes. All other, nonrepresented nodes will also be pruned.

Search tree node 0: The algorithm starts with an empty permutation $\Pi_0 = \langle \rangle$, LB = 0, and $C^* = \emptyset$. The pruning strategy is not applied. Algorithm *getCandidates* computes the candidate list $CL = \{1, 2, 3, 4, 5, 6, 7, 8\}$ formed by all vertices in line 13. The lower bound is improved and updated in line 18 to LB = 1. Any node may be randomly selected from the candidate list CL in line 19, for example, v = 1. A new γ -clique $C^* = \{1\}$ is obtained in line 20. The loop in lines 22–25 will extend the current sequence by appending to Π_0 each node of the candidate list. Algorithm *QClique* will be recursively applied to each sequence $\Pi_1 = \Pi_0 \oplus w$, for every $w \in CL = \{1, 2, 3, 4, 5, 6, 7, 8\}$.

Search tree node 1: Algorithm *QClique* is now recursively applied for the first time to $\Pi_1 = <1 >$. The pruning strategy is applied for the first time and k is set to 2 in line 3. Sequences $\Pi' = <2 >$ and $\Pi'' = <2 >$ are built in lines 4 and 7, respectively. Since inequality (22) is valid for j = 1, the node is not pruned. Next, algorithm *getCandidates* computes the new candidate list in line 13, which is now $CL = \{2, 3, 4, 6, 7\}$. The lower bound is improved and updated in line 18 to LB = 2. Any node may be randomly selected from the candidate list CL in line 19, for example, v = 2. A new γ -clique $C^* = \{1, 2\}$ is obtained in line 20. The loop in lines 22–25 will extend the current sequence by appending to Π_1 each node in the candidate list. Algorithm *QClique* will be recursively applied to each sequence $\Pi_2 = \Pi_1 \oplus w$, for every $w \in CL = \{2, 3, 4, 6, 7\}$.

Search tree node 2: The algorithm is now recursively applied to $\Pi_2 = <1, 2>$. The pruning strategy is applied and k is set to 3 in line 3. Sequences $\Pi' = <3>$ and $\Pi'' = <3>$ are built in lines 4 and 7, respectively. Since inequality (22) is valid for j = 1, the node is not pruned. Next, algorithm *getCandidates* computes the new candidate list $CL = \{3\}$ in line 13. The lower bound is improved and updated in line 18 to LB = 3. Node v = 3 is selected as the only element in the candidate list CL in line 19. A new γ -clique $C^* = \{1, 2, 3\}$ is obtained in line 20. The loop in lines 22–25 will extend

© 2019 The Authors.

the current sequence by appending to Π_2 the unique node in the candidate list. Algorithm *QClique* will be recursively applied to each sequence $\Pi_3 = \Pi_2 \oplus w$, for every $w \in CL = \{3\}$.

Search tree node 3: Algorithm *QClique* is now recursively applied to $\Pi_3 = < 1, 2, 3 >$. The pruning strategy is applied and k is set to 4 in line 3. Sequences $\Pi' = < 4 >$ and $\Pi'' = < 4 >$ are built in lines 4 and 7, respectively. Since inequality (22) is valid for j = 1, the node is not pruned. Next, algorithm *getCandidates* computes the new candidate list $CL = \{4, 5\}$ in line 13. The lower bound is improved and updated in line 18 to LB = 4. Any node may be randomly selected from the candidate list CL in line 19, for example, v = 4. A new γ -clique $C^* = \{1, 2, 3, 4\}$ is obtained in line 20. The loop in lines 22–25 will extend the current sequence by appending to Π_3 each node in the candidate list. Algorithm *QClique* will be recursively applied to each sequence $\Pi_4 = \Pi_3 \oplus w$, for every $w \in CL = \{4, 5\}$.

Search tree node 4: The recursive algorithm is now applied to $\Pi_4 = <1, 2, 3, 4 >$. The pruning strategy is applied and k is set to 5 in line 3. Sequences $\Pi' = <5 >$ and $\Pi'' = <5 >$ are built in lines 4 and 7, respectively. Inequality (22) for j = 1 becomes $|E(\Pi_4)| + deg_G(f_1 = 5, \Pi_4) + \min\{\frac{deg_G(f_1'=5)-deg_G(f_1=5,\Pi_4)}{2}, \binom{1}{2}\} = 5 + 2 + \min\{\frac{3-2}{2}, 0\} = 7 \ge \gamma \cdot \binom{5}{2} = 8$. Since it is not valid, this node is pruned.

Search tree node 5: Algorithm *QClique* is recursively applied to $\Pi_4 = < 1, 2, 3, 5 >$. The pruning strategy is applied and k is set to 5 in line 3. Sequences $\Pi' = <4 >$ and $\Pi'' = <4 >$ are built in lines 4 and 7, respectively. Inequality (22) for j = 1 becomes $|E(\Pi_4)| + deg_G(f_1 = 4, \Pi_4) + \min\{\frac{deg_G(f_1'=4)-deg_G(f_1=4,\Pi_4)}{2}, \binom{1}{2}\} = 5 + 2 + \min\{\frac{4-2}{2}, 0\} = 7 \ge \gamma \cdot \binom{5}{2} = 8$. Since it is not valid, this node is pruned.

Search tree node 6: Algorithm *QClique* is recursively applied to $\Pi_2 = < 1, 3 >$. We recall that at this point LB = 4 and $C^* = \{1, 2, 3, 4\}$. The pruning strategy is applied and k becomes LB + 1 = 5 in line 3. Sequences $\Pi' = < 2, 4, 5 >$ and $\Pi'' = < 2, 4, 5 >$ are built in lines 4 and 7, respectively. Since inequality (22) is valid for j = 1, 2, 3, the node is not pruned. Next, algorithm *getCandidates* computes the new candidate list $CL = \{4\}$ in line 13, formed by only one node. The lower bound *LB* is not improved. The loop in lines 22–25 will extend the current sequence by appending to Π_2 the unique node in the candidate list. Algorithm *QClique* will be recursively applied to each sequence $\Pi_3 = \Pi_2 \oplus w$, for every $w \in CL = \{4\}$.

Search tree node 7: The recursive algorithm is now applied to $\Pi_3 = <1, 3, 4>$. The pruning strategy is applied and k is set to 5 in line 3. Sequences $\Pi' = <2, 6>$ and $\Pi'' = <2, 5>$ are built in lines 4 and 7, respectively. Since inequality (22) is valid for j = 1, 2, the node is not pruned. Next, algorithm *getCandidates* computes the new candidate list $CL = \{6, 7\}$ in line 13. The loop in lines 22–25 will extend the current sequence by appending to Π_3 each node in the candidate list. Algorithm *QClique* will be recursively applied to each sequence $\Pi_4 = \Pi_3 \oplus w$, for every $w \in CL = \{6, 7\}$.

Search tree node 8: The recursive algorithm is now applied to $\Pi_4 = < 1, 3, 4, 6 >$. The pruning strategy is applied and k is set to 5 in line 3. Sequences $\Pi' = < 7 >$ and $\Pi'' = < 2 >$ are built in lines 4 and 7, respectively. Since inequality (22) is valid for j = 1, the node is not pruned. Next, algorithm *getCandidates* computes the new candidate list $CL = \emptyset$ in line 13 and the algorithm returns in line 15.

Search tree node 9: The recursive algorithm is now applied to $\Pi_4 = < 1, 3, 4, 7 >$. The pruning strategy is applied and k is set to 5 in line 3. Sequences $\Pi' = < 6 >$ and $\Pi'' = < 2 >$ are built

19

International Transactions in Operational Research © 2019 International Federation of Operational Research Societies

in lines 4 and 7, respectively. Since inequality (22) is valid for j = 1, once again the node is not pruned. Algorithm *getCandidates* computes the candidate list $CL = \emptyset$ in line 13 and, once again, the algorithm returns in line 15.

Search tree node 10: Algorithm *QClique* is recursively applied to $\Pi_2 = < 1, 4 >$. We recall that still LB = 4 and $C^* = \{1, 2, 3, 4\}$. The pruning strategy is applied and k becomes LB + 1 = 5 in line 3. Sequences $\Pi' = < 3, 6, 7 >$ and $\Pi'' = < 2, 3, 5 >$ are built in lines 4 and 7, respectively. Since inequality (22) is valid for j = 1, 2, 3, the node is not pruned. Next, algorithm *getCandidates* computes the new candidate list $CL = \{6, 7\}$ in line 13. The lower bound *LB* is not improved. The loop in lines 22–25 will extend the current sequence by appending to Π_2 each node in the candidate list. Algorithm *QClique* will be recursively applied to each sequence $\Pi_3 = \Pi_2 \oplus w$, for every $w \in CL = \{6, 7\}$.

Search tree node 11: Algorithm *QClique* is recursively applied to $\Pi_3 = < 1, 4, 6 >$. The pruning strategy is applied and *k* is set to 5 in line 3. Sequences $\Pi' = < 3, 7 >$ and $\Pi'' = < 2, 3 >$ are built in lines 4 and 7, respectively. Since inequality (22) is valid for j = 1, 2, the node is not pruned. Next, algorithm *getCandidates* computes the new candidate list $CL = \{7\}$ in line 13. The lower bound *LB* is not improved. The loop in lines 22–25 will extend the current sequence by appending to Π_3 the unique node in the candidate list. Algorithm *QClique* will be recursively applied to each sequence $\Pi_4 = \Pi_3 \oplus w$, for every $w \in CL = \{7\}$.

Search tree node 12: Algorithm *QClique* is recursively applied to $\Pi_4 = < 1, 4, 6, 7 >$. The pruning strategy is applied and *k* is set to 5 in line 3. Sequences $\Pi' = < 3 >$ and $\Pi'' = < 2 >$ are built in lines 4 and 7, respectively. Since inequality (22) is valid for j = 1, the node is not pruned. Next, algorithm *getCandidates* computes the new candidate list $CL = \{3\}$ in line 13. The lower bound is improved and updated in line 18 to LB = 5. Node 3 is selected as the only element in the candidate list CL in line 19. A new γ -clique $C^* = \{1, 4, 6, 7, 3\}$ is obtained in line 20. The loop in lines 22–25 will extend the current sequence by appending to Π_4 the unique node in the candidate list. Algorithm *QClique* will be recursively applied to $\Pi_5 = \Pi_4 \oplus 3$.

Search tree node 13: Algorithm *QClique* is recursively applied to $\Pi_5 = <1, 4, 6, 7, 3 >$. The pruning strategy is applied and k is set to LB + 1 = 6 in line 3. Sequences $\Pi' = <2 >$ and $\Pi'' = <2 >$ are built in lines 4 and 7, respectively. Inequality (22) for j = 1 becomes $|E(\Pi_5)| + deg_G(f_1 = 2, \Pi_5) + \min\{\frac{deg_G(f_1'=2) - deg_G(f_1=2, \Pi_5)}{2}, \binom{3}{2}\} = 8 + 2 + \min\{\frac{4-2}{2}, 0\} = 10 \ge \gamma \cdot \binom{6}{2} = 12$. Since it is not valid, this node is pruned.

Search tree node 14: Algorithm *QClique* is recursively applied to $\Pi_3 = <1, 4, 7 >$. The pruning strategy is applied and *k* is set to LB + 1 = 6 in line 3, with LB = 5 and $C^* = \{1, 4, 6, 7, 3\}$. Sequences $\Pi' = <6, 3, 2 >$ and $\Pi'' = <2, 3, 5 >$ are built in lines 4 and 7, respectively. Inequality (22) for j = 3 becomes $|E(\Pi_i)| + \sum_{p=1}^{3} deg_G(f_p, \Pi_i) + \min\{\frac{\sum_{p=1}^{3} deg_G(f_p') - \sum_{p=1}^{3} deg_G(f_p, \Pi_i), \binom{3}{2}\} = |E(\Pi_3)| + deg_G(f_1 = 6, \Pi_3) + deg_G(f_2 = 3, \Pi_3) + deg_G(f_3 = 2, \Pi_3) + \min\{(deg_G(f_1' = 2) + deg_G(f_2' = 3) + deg_G(f_3' = 3) - deg_G(f_1 = 6, \Pi_3) - deg_G(f_2 = 3, \Pi_3) - deg_G(f_3 = 2, \Pi_3))/2, 3\} = 3 + (3 + 2 + 1) + \min\{(4 + 4 + 3 - 3 - 2 - 1)/2, 3\} = 11.5 \ge \gamma \cdot \binom{i+j}{2} = \gamma \cdot \binom{3+3}{2} = 12$. Since it is not valid, this node is pruned.

Search tree node 15: Algorithm *QClique* is recursively applied to $\Pi_2 = <1, 6>$. The pruning strategy is applied and k is set to LB + 1 = 6 in line 3. Sequences $\Pi' = <4, 7, 3, 2>$ and $\Pi'' = <2, 3, 4, 5>$ are built in lines 4 and 7, respectively. Also in this case, inequality (22) is not valid for j = 4. Consequently, this node is pruned.

© 2019 The Authors.

Search tree node 16: Algorithm *QClique* is recursively applied to $\Pi_2 = <1, 7>$. The pruning strategy is applied and k is set to LB + 1 = 6 in line 3. Sequences $\Pi' = <4, 6, 2, 3>$ and $\Pi'' = <2, 3, 4, 5>$ are built in lines 4 and 7, respectively. Also in this case, inequality (22) is not valid for j = 4. Consequently, this node is pruned.

Search tree node 17: Algorithm *QClique* is recursively applied to $\Pi_1 = <2>$. The pruning strategy is applied and k is set to LB + 1 = 6 in line 3. Sequences $\Pi' = <1, 3, 5, 8, 4>$ and $\Pi'' = <1, 3, 4, 5, 6>$ are built in lines 4 and 7, respectively. Also in this case, inequality (22) is not valid for j = 5. Consequently, this node is pruned.

The recursion of algorithm *QClique* proceeds following the same strategy and prunes all other nodes of the search tree, returning the optimal γ -clique $C^* = \{1, 4, 6, 7, 3\}$ of maximum cardinality 5.

6. Computational experiments

All algorithms have been implemented using version 19.00.23504 of the Microsoft C/C++ Optimizing compiler. The computational experiments have been performed on an Intel Core i5-5200 processor with 2.20 GHz and 8 GB of RAM running under Windows 10. The test problems comprise 50 randomly generated graphs, 16 instances derived from maximum clique problems of the Second DIMACS Implementation Challenge (Johnson, 1996; DIMACS, 2016) reported in Pajouh et al. (2014), 16 real-life sparse graph instances obtained from the University of Florida Sparse Matrix Collection (Davis and Hu, 2011), and the graph "homer" from Trick (2002).

The exact *QClique* algorithm is compared with the two best formulations in Pattillo et al. (2013) and Veremyev et al. (2016) solved by CPLEX and with the branch-and-bound algorithm in Pajouh et al. (2014). The MIP formulations were solved with version 12.6.2 of the CPLEX library for Visual Studio 2010. The solver was used with the default settings for preprocessing, branching strategies, node algorithms, heuristics, and cutting planes. The upper bound proposed in Section 4 was used in the algorithm of Pajouh et al. (2014) and in model F3.

6.1. Experiments on randomly generated graphs with 100 vertices

We have randomly generated ten instances with 100 vertices each, for each density $\rho = 0.05$, 0.10, 0.25, 0.50, and 0.75. Each instance was solved by the branch-and-bound algorithm (B&B), by the two MIP models (F1 and F3) and by the proposed *QClique* algorithm. The time limit for each algorithm and instance was set to one hour.

The numerical results are presented in Table 1. Whenever any of the four compared exact methods failed to prove optimality within the time limit for all instances in the group of same density, we indicate it by ">3600 (0)." Otherwise, for each group of same density instances, the table shows the number of instances (#opt) solved to optimality within the time limit, the average running time in seconds over the instances solved to proved optimality, and the average size of the best solution found. In addition, for the proposed *QClique* algorithm, we also present the average gap in percentage: for each instance, the relative gap is computed as the difference between the upper bound and the best solution value, divided by the upper bound. The best average times and the best average solution sizes are highlighted in boldface.

Table 1

Average results for randomly generated graphs with 100 vertices (10 instances for each threshold γ and each density ρ), times in seconds

		B&B		F1		F3		QClique		
ρ	γ	Time (#opt)	Avg.	Time (#opt)	Avg.	Time (#opt)	Avg.	Time (#opt)	Avg.	Gap (%)
0.05	1	475.92 (9)	4.1	0.07 (10)	4.1	0.12 (10)	4.1	< 0.01 (10)	4.1	0.00
	0.95	475.55 (9)	4.1	38.31 (10)	4.1	0.14 (10)	4.1	< 0.01 (10)	4.1	0.00
	0.90	2014.34 (9)	4.9	42.65 (10)	4.9	0.15 (10)	4.9	< 0.01 (10)	4.9	0.00
	0.85	1517.32 (8)	5.2	51.04 (10)	5.2	0.14 (10)	5.2	< 0.01 (10)	5.2	0.00
0.10	1	2872.42 (9)	5.0	0.09 (10)	5.0	1.30 (10)	5.0	< 0.01 (10)	5.0	0.00
	0.95	2863.03 (9)	5.0	122.70 (10)	5.0	2.58 (10)	5.0	< 0.01 (10)	5.0	0.00
	0.90	3149.90(1)	6.6	213.80 (10)	6.6	1.71 (10)	6.6	< 0.01 (10)	6.6	0.00
	0.85	>3600 (0)	7.4	352.50 (10)	7.5	1.80 (10)	7.5	0.02 (10)	7.5	0.00
0.25	1	>3600 (0)	6.8	0.54 (10)	6.8	>3600 (0)	6.3	0.08 (10)	6.8	0.00
	0.95	>3600(0)	7.4	510.77 (10)	7.5	>3600 (0)	6.2	0.07 (10)	7.5	0.00
	0.90	>3600(0)	9.0	1725.95 (10)	9.2	>3600 (0)	8.2	0.26 (10)	9.2	0.00
	0.85	>3600(0)	10.1	>3600 (0)	10.0	>3600 (0)	9.5	1.97 (10)	10.7	0.00
0.50	1	>3600(0)	9.0	18.19 (10)	9.1	>3600 (0)	6.7	1.21 (10)	9.1	0.00
	0.95	>3600 (0)	10.1	>3600(0)	10.1	>3600 (0)	7.2	1.65 (10)	10.7	0.00
	0.90	>3600(0)	12.1	>3600(0)	11.0	>3600 (0)	8.2	33.68 (10)	13.0	0.00
	0.85	>3600 (0)	14.7	>3600 (0)	13.3	>3600 (0)	9.9	878.88 (10)	16.2	0.00
0.75	1	>3600(0)	15.3	53.10 (10)	16.8	>3600 (0)	8.2	1590.30 (10)	16.8	0.00
	0.95	>3600 (0)	21.7	>3600(0)	43.60	>3600 (0)	9.9	>3600(0)	23.4	70.75
	0.90	>3600(0)	31.0	>3600(0)	33.80	>3600 (0)	17.0	>3600(0)	32.4	61.48
	0.85	>3600(0)	46.1	>3600(0)	28.04	>3600 (0)	32.7	>3600(0)	46.3	47.98

The numerical results for randomly generated graphs show that algorithm *QClique* is much faster than the others and obtains same quality solutions for the instances with small densities. For the problems with larger densities, algorithm *QClique* outperformed the other approaches finding larger γ -cliques in smaller computation times except for $\rho = 0.75$, when model F1 obtained smaller average times for the same average solution quality for $\gamma = 1$.

6.2. Experiments on literature instances

We consider in this section numerical experiments on 16 instances derived from maximum clique problems of the Second DIMACS Implementation Challenge (Johnson, 1996; DIMACS, 2016) reported in Pajouh et al. (2014).

The numbers of vertices and edges, as well as the density of each instance, appear in Table 2, together with the new upper bound *newUB* given by Equation (23) in Section 4.3 and the upper bound ω^u given by Equation (9) (Pattillo et al., 2013). The best upper bounds for each instance are highlighted in boldface. The new pruning upper bound *newUB* is shown to be significantly tighter (i.e., smaller) for all instances.

Numerical results for these instances are displayed in Tables 3 and 4. Each instance was solved for $\gamma = 1, 0.95, 0.90, 0.85$, and 0.80. The time limit for each algorithm and instance was set to

© 2019 The Authors.

		7 <i>E</i>		$\gamma = 1$	$\gamma = 1$		$\gamma = 0.95$		85
Instance	V		ρ	$\overline{\omega^u}$	new UB	ω^u	newUB	$\overline{\omega^u}$	newUB
c-fat200-1	200	1534	0.08	53	18	54	18	57	21
c-fat200-2	200	3235	0.16	79	34	81	36	86	40
c-fat200-5	200	8473	0.43	130	86	133	91	141	101
hamming6-4	64	704	0.35	37	23	38	24	40	26
hamming8-4	256	20,864	0.64	204	164	209	172	221	192
johnson8-2-4	28	210	0.56	20	16	21	16	22	18
johnson8-4-4	70	1855	0.77	61	54	62	56	66	63
brock200-2	200	9876	0.50	140	104	144	110	152	122
brock200-1	200	14,834	0.75	172	151	177	159	187	177
brock200-3	200	12,048	0.61	155	125	159	131	168	146
brock200-4	200	13,089	0.66	162	135	166	141	175	157
brock400-2	400	59,786	0.75	346	303	355	318	375	354
brock400-4	400	59,765	0.75	346	303	355	318	375	354
keller4	171	9435	0.65	137	115	141	120	149	133
p-hat300-1	300	10,933	0.24	147	100	151	104	159	115
p-hat300-2	300	21,928	0.49	209	177	214	185	227	201

Literature graphs: new pruning upper bound UB is significantly tighter

Table 2

one hour. For each of the algorithms employed to solve them (B&B, F1, F3, and *QClique*), the tables show the best solution value and the running time in seconds. In addition, for the proposed *QClique* algorithm, we also present the relative gap, computed as the difference between the upper bound and the best solution value, divided by the upper bound. Whenever any of the four compared exact methods failed to prove optimality within the time limit, we indicate it by "> 3600." The best solution values and the best running times for each instance are highlighted in boldface.

Although B&B performed well for dense instances such as brock400-2 and brock400-4, Veremyev et al. (2016) observed that it performs badly for sparse instances. In general, algorithm *QClique* showed the best performance overall (best solutions and significantly smaller running times in some cases), in particular for instances such as hamming6-4, johnson8-2-4, brock200-2, brock200-3, keller4, and p-hat300-1.

6.3. Experiments on large sparse instances

We consider here 16 real-life sparse graph instances from the University of Florida Sparse Matrix Collection (Davis and Hu, 2011), as well as the graph "homer" from (Trick, 2002). We compare algorithm *QClique* with the results obtained by model F3, which is the best adapted for sparse graphs among those in the literature. Each instance was solved for γ ranging from 0.60 to 1 by increments of 0.05. The time limit for each algorithm and instance was set to one hour. For each instance, Tables 5–7 present its description, the value of the threshold γ , and, for each approach employed to solve it (F3 and *Qclique*), the best solution value and the running time in seconds. In addition, for the proposed *QClique* algorithm, we also present the relative gap, computed as the difference between the upper bound and the best solution value, divided by the upper bound.

Table 3

Comparative results for B&B, F1, F3, and *QClique* on literature instances for $\gamma = 1, 0.95, 0.90, 0.85$, and 0.80, times in seconds—Part I

		B&B		F1		F3		QCliq	ие	
Instance	γ	Best	Time	Best	Time	Best	Time	Best	Time	Gap (%)
c-fat200-1	1	12	>3600	12	13.11	12	5.95	12	0.05	0.00
	0.95	12	>3600	12	736.98	12	17.47	12	0.53	0.00
	0.90	13	>3600	13	1210.06	13	10.33	13	2.39	0.00
	0.85	14	>3600	14	1310.38	14	9.38	14	14.33	0.00
	0.80	16	>3600	16	3554.00	16	5.80	16	87.56	0.00
c-fat200-2	1	24	>3600	24	11.00	24	369.17	24	2.23	0.00
	0.95	25	>3600	25	1142.48	25	197.55	25	1397.16	0.00
	0.90	25	>3600	26	1988.50	26	249.56	26	>3600	31.58
	0.85	29	>3600	29	3088.20	29	143.00	29	>3600	27.50
	0.80	32	>3600	32	>3600	32	142.16	32	>3600	23.81
c-fat200-5	1	58	>3600	58	32.05	58	>3600	58	>3600	32.56
	0.95	61	>3600	61	2079.05	61	>3600	61	>3600	32.97
	0.90	65	>3600	63	>3600	62	>3600	65	>3600	32.29
	0.85	70	>3600	68	>3600	70	>3600	70	>3600	30.69
	0.80	79	>3600	78	>3600	78	>3600	74	>3600	30.84
hamming6-4	1	4	15.90	4	0.53	4	13.81	4	< 0.01	0.00
	0.95	4	15.44	4	18.27	4	571.94	4	0.02	0.00
	0.90	4	16.01	4	23.55	4	996.11	4	0.03	0.00
	0.85	4	15.71	4	26.09	4	180.83	4	< 0.01	0.00
	0.80	6	891.47	6	36.45	6	955.95	6	0.11	0.00
hamming8-4	1	16	>3600	16	30.84	5	>3600	16	>3600	90.24
	0.95	17	>3600	7	>3600	4	>3600	17	>3600	90.12
	0.90	19	>3600	17	>3600	3	>3600	21	>3600	88.46
	0.85	35	>3600	11	>3600	8	>3600	28	>3600	85.42
	0.80	62	>3600	15	>3600	5	>3600	57	>3600	72.06
johnson8-2-4	1	4	296.07	4	0.02	4	0.19	4	0.02	0.00
	0.95	4	0.02	4	1.39	4	0.25	4	< 0.01	0.00
	0.90	4	0.11	4	1.78	4	0.20	4	< 0.01	0.00
	0.85	4	1.40	4	2.34	4	0.22	4	< 0.01	0.00
	0.80	5	0.36	5	2.55	5	0.86	5	0.03	0.00
johnson8-4-4	1	10	>3600	14	0.14	7	>3600	14	12.78	0.00
	0.95	12	>3600	15	126.09	10	>3600	15	195.08	0.00
	0.90	13	>3600	17	>3600	14	>3600	17	>3600	71.17
	0.85	21	>3600	22	>3600	21	>3600	25	>3600	60.32
	0.80	24	>3600	39	>3600	36	>3600	41	>3600	38.81
brock200-2	1	9	>3600	12	1315.750	5	>3600	12	38.50	0.00
	0.95	10	>3600	12	>3600	5	>3600	13	112.67	0.00
	0.90	14	>3600	12	>3600	5	>3600	16	>3600	86.09
	0.85	15	>3600	15	>3600	6	>3600	18	>3600	85.25
	0.80	20	>3600	17	>3600	9	>3600	23	>3600	82.17

© 2019 The Authors.

Table 4

Comparative results for B&B, F1, F3, and *QClique* on literature instances for $\gamma = 1, 0.95, 0.90, 0.85$, and 0.80, times in seconds—Part II

		B&B		F1		F3		QCliqı	ие	
Instance	γ	Best	Time	Best	Time	Best	Time	Best	Time	Gap
brock200-1	1	15	>3600	19	>3600	4	>3600	19	>3600	87.42
	0.95	26	>3600	22	>3600	8	>3600	27	>3600	83.02
	0.90	42	>3600	27	>3600	11	>3600	39	>3600	76.65
	0.85	63	>3600	22	>3600	13	>3600	62	>3600	64.97
	0.80	112	>3600	108	>3600	69	>3600	106	>3600	43.32
brock200-3	1	10	>3600	15	3586.77	5	>3600	15	>3600	88.00
	0.95	15	>3600	14	>3600	7	>3600	18	>3600	86.26
	0.90	21	>3600	17	>3600	7	>3600	21	>3600	84.78
	0.85	25	>3600	22	>3600	12	>3600	27	>3600	81.51
	0.80	37	>3600	29	>3600	17	>3600	36	>3600	76.62
brock200-4	1	12	>3600	15	>3600	6	>3600	16	>3600	88.15
	0.95	18	>3600	21	>3600	5	>3600	20	>3600	85.82
	0.90	24	>3600	20	>3600	5	>3600	25	>3600	83.22
	0.85	34	>3600	26	>3600	14	>3600	33	>3600	78.98
	0.80	51	>3600	23	>3600	22	>3600	51	>3600	69.28
brock400-2	1	18	>3600	15	>3600	1	>3600	23	>3600	92.41
	0.95	34	>3600	0	>3600	1	>3600	31	>3600	90.25
	0.90	54	>3600	0	>3600	1	>3600	49	>3600	85.37
	0.85	94	>3600	0	>3600	1	>3600	82	>3600	76.84
	0.80	180	>3600	0	>3600	2	>3600	169	>3600	54.93
brock400-4	1	18	>3600	17	>3600	1	>3600	23	>3600	92.41
	0.95	32	>3600	8	>3600	3	>3600	30	>3600	90.57
	0.90	53	>3600	0	>3600	3	>3600	47	>3600	85.97
	0.85	93	>3600	0	>3600	1	>3600	88	>3600	75.14
	0.80	183	>3600	0	>3600	1	>3600	171	>3600	54.40
keller4	1	8	>3600	11	1295.20	4	>3600	11	475.38	0.00
	0.95	10	>3600	14	>3600	5	>3600	15	>3600	87.50
	0.90	16	>3600	17	>3600	7	>3600	21	>3600	83.33
	0.85	23	>3600	19	>3600	9	>3600	24	>3600	81.96
	0.80	37	>3600	40	>3600	17	>3600	44	>3600	68.79
p-hat300-1	1	8	>3600	8	2077.28	6	>3600	8	3.24	0.00
	0.95	8	>3600	7	>3600	4	>3600	9	3.53	0.00
	0.90	10	>3600	9	>3600	5	>3600	11	30.56	0.00
	0.85	11	>3600	10	>3600	6	>3600	12	358.16	0.00
	0.80	13	>3600	11	>3600	8	>3600	15	>3600	87.60
p-hat300-2	1	23	>3600	24	>3600	5	>3600	25	>3600	85.88
	0.95	39	>3600	28	>3600	18	>3600	33	>3600	82.16
	0.90	60	>3600	44	>3600	8	>3600	59	>3600	69.43
	0.85	85	>3600	69	>3600	5	>3600	84	>3600	58.21
	0.80	114	>3600	78	>3600	75	>3600	113	>3600	46.45

Table 5 Comparative results for F3 and QClique on large sparse instances—	Part I
F3	OCliaue

		F3		QClique		
Instance	γ	Best	Time	Best	Time	Gap (%)
Erdos02	1	7	38.97	7	1.72	0.00
V = 6927	0.95	7	56.72	7	1.83	0.00
E = 8472	0.90	8	39.64	8	1.87	0.00
$\rho = 0.0004$	0.85	9	113.77	9	2.30	0.00
	0.80	10	27.70	10	6.58	0.00
	0.75	10	38.45	10	14.92	0.00
	0.70	11	23.28	11	134.52	0.00
	0.65	13	17.53	13	2437.39	0.00
	0.60	14	12.27	13	>3600	83.54
Erdos971	1	7	31.91	7	0.03	0.00
V = 472	0.95	7	51.13	7	0.03	0.00
E = 1314	0.90	8	39.89	8	0.05	0.00
$\rho = 0.0118$	0.85	9	14.00	9	0.10	0.00
	0.80	11	6.88	11	0.65	0.00
	0.75	12	5.55	12	3.10	0.00
	0.70	13	9.38	13	29.68	0.00
	0.65	15	3.66	15	2020.25	0.00
	0.60	17	3.19	17	>3600	57.50
Erdos972	1	7	8.56	7	1.02	0.00
V = 5488	0.95	7	12.55	7	0.99	0.00
E = 7085	0.90	8	8.02	8	1.08	0.00
$\rho = 0.0005$	0.85	9	4.36	9	1.36	0.00
	0.80	11	2.75	11	3.11	0.00
	0.75	12	2.73	12	10.57	0.00
	0.70	13	2.66	13	211.07	0.00
	0.65	14	3.20	14	>3600	78.46
	0.60	15	2.80	15	>3600	78.26
Erdos992	1	8	9.91	8	1.44	0.00
V = 6100	0.95	8	6.72	8	1.75	0.00
E = 7515	0.90	9	7.06	9	1.75	0.00
$\rho = 0.0004$	0.85	10	4.36	10	1.92	0.00
	0.80	11	4.05	11	5.32	0.00
	0.75	12	2.77	12	34.49	0.00
	0.70	13	2.55	13	426.79	0.00
	0.65	14	3.75	14	>3600	78.79
	0.60	16	3.09	16	>3600	77.14
Eva	1	4	2.42	4	1.85	0.00
V = 8497	0.95	4	2.47	4	1.64	0.00
E = 6711	0.90	4	2.63	4	1.63	0.00
$\rho = 0.0002$	0.85	4	4.00	4	1.54	0.00
	0.80	5	1.82	5	1.70	0.00
	0.75	5	2.27	5	1.66	0.00
	0.70	5	2.27	5	1.54	0.00
	0.65	6	2.19	6	36.92	0.00
	0.60	7	1.80	7	39.81	0.00

26

		F3		QClique		
Instance	γ	Best	Time	Best	Time	Gap (%)
California	1	3	>3600	10	3.99	0.00
V = 9664	0.95	2	>3600	11	4.64	0.00
E = 15,969	0.90	2	>3600	12	5.09	0.00
$\rho = 0.0003$	0.85	8	>3600	13	5.45	0.00
	0.80	13	>3600	13	44.81	0.00
	0.75	2	>3600	14	1096.36	0.00
	0.70	3	>3600	14	3024.54	0.00
	0.65	7	>3600	15	>3600	84.69
	0.60	15	>3600	13	>3600	87.50
homer	1	13	18.81	13	0.08	0.00
V = 561	0.95	13	11.39	13	0.31	0.00
E = 1628	0.90	14	9.14	14	1.56	0.00
$\rho = 0.0104$	0.85	16	2.72	16	16.23	0.00
	0.80	18	2.05	18	520.29	0.00
	0.75	20	1.20	20	>3600	55.56
	0.70	22	1.66	22	>3600	53.19
	0.65	24	1.45	24	>3600	51.02
	0.60	27	1.13	27	>3600	48.08
SmallW	1	7	28.45	7	0.06	0.00
V = 396	0.95	8	6.45	8	0.06	0.00
E = 994	0.90	11	2.73	11	0.28	0.00
$\rho = 0.0127$	0.85	12	1.88	12	1.72	0.00
	0.80	14	1.50	14	26.78	0.00
	0.75	16	0.97	16	598.68	0.00
	0.70	17	0.83	17	>3600	57.50
	0.65	19	0.66	19	>3600	54.76
	0.60	22	0.66	20	>3600	54.55
email	1	7	>3600	12	0.19	0.00
V = 1133	0.95	10	>3600	12	0.27	0.00
E = 5451	0.90	13	>3600	13	0.53	0.00
$\rho = 0.0085$	0.85	14	>3600	14	1.52	0.00
	0.80	12	>3600	15	15.31	0.00
	0.75	16	>3600	16	127.70	0.00
	0.70	17	>3600	17	2258.96	0.00
	0.65	18	1993.52	17	>3600	70.18
	0.60	20	2099.95	19	>3600	68.85
USAir97	1	21	>3600	22	727.24	0.00
V = 332	0.95	29	176.36	29	>3600	46.30
E = 2126	0.90	35	3.48	35	>3600	37.50
$\rho = 0.0387$	0.85	39	1.19	39	>3600	32.76
	0.80	43	0.72	43	>3600	28.33
	0.75	46	0.52	46	>3600	25.81
	0.70	49	0.20	49	>3600	24.62
	0.65	53	0.20	53	>3600	22.06
	0.60	57	0.23	57	>3600	20.83

Comparative results for F3 and *QClique* on large sparse instances-Part II

Table 6

© 2019 The Authors.

able 7 Comparative results for F3 and <i>QClique</i> on large sparse instances—Part III									
		F3		QClique					
Instance	γ	Best	Time	Best					

Instance	γ	Best	Time	Best	Time	Gap (%)
nemeth04	1	1	>3600	22	>3600	6.86
V = 9506	0.95	1	>3600	27	>3600	9.06
E = 192,651	0.90	1	>3600	30	>3600	46.43
$\rho = 0.0043$	0.85	1	>3600	29	>3600	50.85
	0.80	1	>3600	32	>3600	49.21
	0.75	1	>3600	38	>3600	43.28
	0.70	1	>3600	45	>3600	37.50
	0.65	1	>3600	50	>3600	35.07
	0.60	1	>3600	53	>3600	36.91
vsp_p0291_seymourl_iiasa	1	1	>3600	5	12.33	0.00
V = 10,498	0.95	1	>3600	5	12.08	0.00
E = 53,868	0.90	1	>3600	6	12.28	0.00
$\rho = 0.0010$	0.85	1	>3600	7	12.55	0.00
	0.80	1	>3600	7	17.64	0.00
	0.75	1	>3600	9	23.10	0.00
	0.70	1	>3600	10	153.04	0.00
	0.65	1	>3600	11	2951.08	0.00
	0.60	1	>3600	12	>3600	93.37

Whenever any of the two compared exact methods failed to prove optimality within the time limit, we indicate it by "> 3600." The best solution values and the best running times for each instance are highlighted in boldface.

The numerical results show that algorithm *QClique* performs significantly better than F3 for ten out of the 16 instances in this class, in particular for large values of the threshold γ . Although for the remaining six instances (USAir97, Harvard500, Geom, PGPgiantcompo, ca-HepTh, and netscience) F3 is faster, we observe that for some instances (such as nemeth04 and vsp_p0291_seymourl_iiasa) F3 was not even capable to find the linear relaxation at the root node of the search tree within the time limit of one hour.

6.4. Evaluating the effect of pruning

In this last experiment, we evaluate the effectiveness of the pruning procedure developed in Section 3, showing by how much algorithm *QClique* improves the efficiency of the naïve algorithm *Backtrack-ing* presented in Section 3. This evaluation is validated on eight of the large sparse instances in Tables 5–7 for which algorithm *QClique* finished within the time limit for any value of $\gamma \ge 0.7$.

Each instance was solved for γ ranging from 0.70 to 1 by increments of 0.05. The time limit for each algorithm and instance was set to one hour. For each instance, Tables 8 and 9 present its description, the value of the threshold γ , and, for each approach employed to solve it (*Backtracking* and *Qclique*), the best solution value and the running time in seconds. In addition, for the proposed *QClique* algorithm, we also present the number of cuts applied to prune the search tree. Whenever

© 2019 The Authors.

International Transactions in Operational Research © 2019 International Federation of Operational Research Societies

- - - -

		Backtracking		QClique		
Instance	γ	Time (seconds)	Nodes	Time (seconds)	Nodes	Cuts
Erdos02	1	1.27	18,636	1.72	17,083	16,263
	0.95	1.34	18,643	1.83	17,476	16,368
	0.90	1.55	20,285	1.87	18,486	16,852
	0.85	1.75	23,625	2.30	21,052	16,852
	0.80	6.27	96,781	6.58	66,611	48,785
	0.75	19.82	317,666	14.92	154,269	117,688
	0.70	232.49	4,151,721	134.52	1,639,854	1,441,491
Erdos971	1	0.01	3820	0.03	2917	2233
	0.95	0.01	3841	0.03	3079	2247
	0.90	0.02	5814	0.05	4433	2998
	0.85	0.05	11,391	0.10	8946	4861
	0.80	0.29	78,464	0.66	54,444	33,216
	0.75	1.93	481,838	3.10	234,420	165,279
	0.70	20.71	5,828,475	29.68	2,538,313	1,856,004
Erdos972	1	0.69	14,572	1.02	13,473	12,813
	0.95	0.68	14,593	0.99	13,721	12,858
	0.90	0.74	15,807	1.08	14,753	13,461
	0.85	0.93	19,429	1.36	17,498	14,009
	0.80	2.75	60,345	3.11	41,481	28,889
	0.75	16.65	332,385	10.57	138,303	108,841
	0.70	211.07	5,026,686	98.38	1,489,570	1,278,601
Erdos992	1	1.59	15,911	1.44	14,438	13,802
	0.95	1.06	15,971	1.75	14,754	13,887
	0.90	1.03	17,758	1.75	16,636	14,946
	0.85	1.39	25,174	1.92	21,608	16,593
	0.80	5.61	104,257	5.32	62,601	43,017
	0.75	68.21	1,272,873	34.49	414,903	356,406
	0.70	1511.70	31,238,995	426.79	5,837,764	5,293,928
Eva	1	1.28	15,288	1.85	14,030	13,701
	0.95	1.32	15,288	1.64	14,030	13,701
	0.90	1.22	15,288	1.63	14,344	13,853
	0.85	1.17	15,288	1.54	14,344	13,853
	0.80	1.22	15,376	1.70	14,851	13,766
	0.75	1.20	15,376	1.66	14,851	13,766
	0.70	1.13	15,551	1.54	14,940	13,831
California	1	3.51	30,094	3.99	26,609	25,241
	0.95	3.30	31,010	4.64	28,300	25,420
	0.90	4.36	34,697	5.09	32,336	27,114
	0.85	4.01	39,471	5.45	35,801	27,379
	0.80	100.87	1,086,317	44.81	362,939	341,665
	0.75	>3600	45,261,580	1096.36	9,352,431	9,320,103
	0.70	>3600	44,680,352	3024.54	26,966,927	26,874,710

Effect of pruning in the running times and in the number visited of nodes of the search tree—Part I

Table 8

 $$\cite{C}$$ 2019 The Authors. International Transactions in Operational Research \cite{C} 2019 International Federation of Operational Research Societies

Instance	γ	Backtracking		QClique		
		Time (seconds)	Nodes	Time (seconds)	Nodes	Cuts
email	1	0.17	20,473	0.19	10,205	8562
	0.95	0.18	21,484	0.27	15,189	9258
	0.90	0.32	40,212	0.53	28,705	17,061
	0.85	1.43	162,477	1.52	75,449	44,066
	0.80	28.28	3,490,864	15.31	842,449	690,627
	0.75	575.96	76,742,311	127.70	7,810,753	7,042,802
	0.70	>3600	469,138,315	2258.96	141,058,106	131,416,695
vsp_p0291_seymourl_iiasa	1	9.97	67,844	12.33	66,088	58,436
	0.95	9.64	67,844	12.08	66,088	58,436
	0.90	9.78	68,457	12.28	66,746	58,039
	0.85	9.96	69,171	12.55	68,011	57,097
	0.80	14.90	114,763	17.64	99,398	78,463
	0.75	19.10	155,965	23.10	132,518	99,346
	0.70	150.16	1,323,372	153.04	925,051	812,402

Effect of pruning in the running times and in the number visited of nodes of the search tree-Part II

any of the two compared exact methods failed to prove optimality within the time limit, we indicate it by "> 3600."

The results in these tables show that, although the number of cuts may be very high for some instances, they lead to considerable reductions in the number of nodes of the search tree that are visited. As an example, consider the case of instance Erdos992, when 5,293,928 cuts are generated. Observe that the number of nodes of the search tree that are visited is reduced from 31,238,995 (*Backtracking*) to 5,837,764 (*QClique*), that is, to only 18.68% of the original number. The running time is reduced from 1511.70 to 426.79 seconds, that is, to 28.17% of the original time. We observe that, although this is the typical behavior for most instances, for some of them (such as vsp_p0291_seymourl_iiasa) the reduction in the number of visited nodes is not followed by a reduction in the running time, due to the time taken to compute the cuts and to apply the pruning procedure.

7. Concluding remarks

Given a graph G = (V, E) and a threshold $\gamma \in (0, 1]$, the maximum quasi-clique problem consists in finding a maximum cardinality subset C^* of the vertices in V such that the density of the graph induced in G by C^* is greater than or equal to the threshold γ .

We proposed an exact enumeration algorithm for solving the maximum quasi-clique problem, based on a quasi-hereditary property. This property is the main principle of the strategy used to design the backtracking algorithm that traverses the search tree. This algorithm by itself already obtained good computational results, as shown in Tables 8 and 9.

We also proposed a new upper bound for the problem. This new upper bound is used for pruning the search tree generated by the backtracking algorithm. It is consistently tighter than previously existing bounds and leads to significant reductions in the enumeration tree.

© 2019 The Authors.

International Transactions in Operational Research © 2019 International Federation of Operational Research Societies

Table 9

Numerical results showed that the improved approach derived from the combination of a cutbased pruning procedure with the original backtracking algorithm is competitive with the best integer programming formulations in Pattillo et al. (2013) and Veremyev et al. (2016) solved by CPLEX and with the branch-and-bound algorithm proposed in Pajouh et al. (2014), in terms of both solution quality and running times, in particular for small values of the threshold γ .

Acknowledgments

Work of Jose A. Riveaux was sponsored by a CNPq scholarship. Work of Celso C. Ribeiro was partially supported by CNPq research grants 303958/2015-4 and 425778/2016-9 and by FAPERJ research grant E-26/202.854/20179. This work was also was partially sponsored by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), under Finance Code 001.

References

- Abello, J., Pardalos, P.M., Resende, M.G.C., 1999. On maximum clique problems in very large graphs. In Abello, J.M., Vitter, J.S. (eds) *External Memory Algorithms*. American Mathematical Society, Providence, RI, pp. 119–130.
- Abello, J., Resende, M., Sudarsky, S., 2002. Massive quasi-clique detection. In Abello, J., Vitter, J. (eds) Proceedings of the 5th Latin American Symposium on the Theory of Informatics, Lecture Notes in Computer Science, Vol. 2286. Springer, Berlin, pp. 598–612.
- Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2001. *Introduction to Algorithms* (2nd edn). MIT Press, Cambridge, MA.
- Davis, T.A., Hu, Y., 2011. The University of Florida sparse matrix collection. ACM Transactions on Mathematical Software 38(1), 1:1–1:25.
- DIMACS, 2016. Dimacs Implementation Challenges. Available at http://archive.dimacs.rutgers.edu/Challenges/ (accessed 18 December 2018).
- Johnson, D.S., 1996. Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26. American Mathematical Society, Providence, RI.
- Karp, R.M., 1972. Reducibility among combinatorial problems. In Miller, R.E., Thatcher, J.W. (eds) Complexity of Computer Computations. Plenum Press, New York, pp. 85–103.
- Nemhauser, G.L., Wolsey, L.A., 1988. Integer Programming and Combinatorial Optimization. Wiley, New York.
- Pajouh, F.M., Miao, Z., Balasundaram, B., 2014. A branch-and-bound approach for maximum quasi-cliques. Annals of Operations Research 216, 145–161.
- Pattillo, J., Veremyev, A., Butenko, S., Boginski, V., 2013. On the maximum quasi-clique problem. *Discrete Applied Mathematics* 161, 244–257.
- Trick, M., 2002. COLOR02/03/04: Graph Coloring and Its Generalizations. Available at http://mat.gsia.cmu.edu/ COLOR03/ (accessed 18 December 2018).
- Veremyev, A., Prokopyev, O.A., Butenko, S., Pasiliao, E.L., 2016. Exact MIP-based approaches for finding maximum quasi-clique and dense subgraph. *Computational Optimization and Applications* 64, 177–214.

APÊNDICE B – A BRKGA-based matheuristic for the maximum quasi-clique problem with an exact local search strategy

A BRKGA-BASED MATHEURISTIC FOR THE MAXIMUM QUASI-CLIQUE PROBLEM WITH AN EXACT LOCAL SEARCH STRATEGY

BRUNO Q. PINTO¹, CELSO C. RIBEIRO^{2,*}, JOSÉ A. RIVEAUX² AND ISABEL ROSSETI²

Abstract. Given a graph G = (V, E) and a threshold $\gamma \in (0, 1]$, the maximum cardinality quasiclique problem consists in finding a maximum cardinality subset C^* of the vertices in V such that the density of the graph induced in G by C^* is greater than or equal to the threshold γ . This problem has a number of applications in data mining, *e.g.*, in social networks or phone call graphs. We propose a matheuristic for solving the maximum cardinality quasi-clique problem, based on the hybridization of a biased random-key genetic algorithm (BRKGA) with an exact local search strategy. The newly proposed approach is compared with a pure biased random-key genetic algorithm, which was the best heuristic in the literature at the time of writing. Computational results show that the hybrid BRKGA outperforms the pure BRKGA.

Mathematics Subject Classification. 05C69, 05C85, 68T20, 90C27, 90C59.

Received June 10, 2019. Accepted January 4, 2020.

1. INTRODUCTION

Let G = (V, E) be a graph defined by a vertex set V and an edge set $E \subseteq V \times V$. G is a complete graph if there is an edge in E connecting every two different vertices in V. A graph G' = (V', E') is a subgraph of G if $V' \subseteq V$ and $E' \subseteq E$, which is denoted by $G' \subseteq G$. The graph G(V') induced in G by $V' \subseteq V$ is that with vertex set V' and edge set formed by all edges of E with both ends in V'. For any $V' \subseteq V$, the subset $E(V') \subseteq E$ is formed by all edges of E with both ends in V'. In other words, E(V') is the edge set of the graph induced in G by V'.

The density of graph G is given by $dens(G) = |E|/(|V| \times (|V| - 1)/2)$. For any $v \in V$, the degree $deg_G(v)$ denotes the number of vertices in G that are adjacent to v. In addition, for any $v \in V$ and any $V' \subseteq V$, we denote by $deg_G(v, V')$ the number of vertices of V' that are adjacent to v in G.

A subset $C \subseteq V$ is a clique of G if the graph G(C) induced in G by C is complete. Given a graph G = (V, E), the maximum clique problem (MCP) consists in finding a maximum cardinality clique of G. It was proved to be NP-hard by [15].

Given a graph G = (V, E) and a threshold $\gamma \in (0, 1]$, a γ -clique is any subset $C \subseteq V$ such that the density of the subgraph G(C) is greater than or equal to γ . A γ -clique C is maximal if there is no other γ -clique C' that

Article published by EDP Sciences

Keywords. Maximum quasi-clique problem, maximum clique problem, biased random-key genetic algorithm, metaheuristics, matheuristics.

¹ Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro, Uberlândia, MG 38411-104, Brazil.

² Institute of Computing, Universidade Federal Fluminense, Niterói, RJ 24210-346, Brazil.

^{*}Corresponding author: celso@ic.uff.br

strictly contains C. The maximum quasi-clique problem (MQCP) amounts to finding a maximum cardinality subset C^* of the vertices in V such that the density of the graph induced in G by C^* is greater than or equal to the threshold γ . This problem was proved to be NP-hard in [21]. The problem has many applications and related clustering approaches include classifying molecular sequences in genome projects by using a linkage graph of their pairwise similarities [8] and the analysis of massive telecommunication data sets obtained from social networks or phone call graphs [1], as well as various other data mining and graph mining applications. The reader is also referred to Pastukhov *et al.* [20] for other examples of real-life applications.

A few heuristics for MQCP exist in the literature, based on well known approaches such as greedy randomized algorithms and their iterated extensions [18], stochastic local search [8], and GRASP [1]. Pinto *et al.* [23] proposed a biased random-key genetic algorithm for finding approximate solutions to the maximum cardinality quasi-clique problem, using two different decoders. The resulting BRKGA-IG^{*} heuristic strategy achieved the best performance and outperformed the restarted optimized iterated greedy (RIG^{*}) construction/destruction heuristic of Oliveira *et al.* [18]. BRKGA-IG^{*} was also compared with the exact algorithms AlgF3 and AlgF4 of Veremyev *et al.* [31] used as heuristics with time limits on their running times. BRKGA-IG^{*} applied to sparse graphs also outperformed the mixed integer programming approaches, finding target solution values in much smaller running times.

Ribeiro and Riveaux [25] proposed the exact enumeration algorithm QClique to solve the maximum quasiclique problem, based on a quasi-hereditary property. They also proposed a new upper bound that is used for pruning the search tree. Numerical results showed that their approach is competitive with the best integer programming formulations in [21, 31] solved by CPLEX and with the branch-and-bound algorithm proposed by Pajouh *et al.* [19], in terms of both solution quality and running time. In general, algorithm QClique showed the best performance on random instances with respect to the best existing MIP models, obtaining the best solutions and significantly smaller running times for several instances. Again, it showed the best performance overall for several literature instances such as hamming6-4, johnson8-2-4, brock200-2, brock200-3, keller4, and p-hat300-1. The numerical results also showed that performed significantly better than AlgF3 for ten out of the 16 large sparse instances, in particular for larger values of the threshold γ . Although for the other six instances AlgF3 was faster, for some instances AlgF3 was not even capable to solve the linear relaxation at the root node of the search tree within the time limit of one hour.

We show that a variant of the exact enumeration algorithm QClique proposed by Ribeiro and Riveaux [25] can be hybridized with the biased random-key genetic algorithm BRKGA-IG^{*} developed by Pinto *et al.* [23] as a local search strategy to improve the quality of the solutions created by the decoder. This paper is organized as follows. Section 2 presents the problem formulation. Section 3 introduces biased random-key genetic algorithms and describes their customization to the maximum quasi-clique problem. Section 4 describes in detail the decoder DECODER-IG^{*} previously used in the implementation of the biased random-key genetic algorithm for the maximum quasi-clique problem. The new decoder DECODER-LSQClique using an exact local search strategy is presented in Section 5, giving rise to a matheuristic for solving the maximum cardinality quasi-clique problem based on a biased random-key genetic algorithm. The exact local search algorithm LSQClique used by the new decoder is described in detail in Section 6. Numerical results are reported in Section 7. Concluding remarks are drawn in the last section.

2. PROBLEM FORMULATION AND RELATED WORK

The maximum quasi-clique problem can be formulated by associating a binary variable x_i to each vertex of the graph [21]:

 $x_i = \begin{cases} 1, & \text{if vertex } v_i \in V \text{ belongs to the solution,} \\ 0, & \text{otherwise.} \end{cases}$

S742

This formulation also considers a variable $y_{ij} = x_i \cdot x_j$ associated to each pair of vertices $i, j \in V$, with i < j, which is linearized as follows:

$$\max \sum_{i \in V} x_i \tag{2.1}$$

subject to:

$$\sum_{[i,j]\in E:i$$

$$y_{ij} \le x_i, \qquad \forall i, j \in V, \qquad i < j, \tag{2.3}$$

$$y_{ij} \le x_j, \qquad \forall i, j \in V, \qquad i < j, \tag{2.4}$$

$$y_{ij} \ge x_i + x_j - 1, \qquad i, j = 1, \dots, n, \quad i < j,$$

 $x_i \in \{0, 1\} \qquad \forall i \in V$
(2.5)

(2.6)

$$\begin{array}{c} & & (2.3) \\ & & & (2.3) \\ & & & (2.3) \\ \end{array}$$

$$y_{ij} \ge 0, \qquad \forall i, j \in V, \qquad i < j.$$

The objective function (2.1) maximizes the number of vertices in the solution. If two vertices i, j belong to a solution, then $x_i = x_j = 1$ and $y_{ij} = x_i \cdot x_j = 1$. If edge $[i, j] \in E$, then it contributes to the density of the quasi-clique. Constraint (2.2) ensures that the density of the solution is greater than or equal to γ . Constraints (2.3) and (2.4) ensure that any edge may contribute to the density of a solution only if both of its ends are chosen to belong to this solution. Constraints (2.5) ensure that any existing edge $[i, j] \in E$ will contribute to the solution if both of its ends are chosen. Constraints (2.6) and (2.7) impose the binary and non-negativity requirements on the problem variables, respectively.

Veremyev *et al.* [31] reported and compared four mixed integer programming formulations for the maximum quasi-clique problem in sparse graphs. Two algorithms based on the best formulations led to better results than the mixed integer programming formulation proposed in [21], with all mixed integer programs solved using FICO Xpress-Optimizer [10] with the time limit of 3600 s.

Ribeiro and Riveaux [25] developed the exact algorithm *QClique* based on a quasi-hereditary proposition and proposed a new upper bound that is used for pruning the search tree. Numerical results showed that their approach is competitive and outperforms the best integer programming approaches in the literature. The new upper bound is consistently tighter than previously existing bounds. The cuts lead to considerable reductions in the number of visited nodes of the search tree. Consider the case of instance Erdos992, when 5 293 928 cuts are generated. The number of visited nodes of the search tree is reduced from 31 238 995 to 5 837 764, *i.e.*, to only 18.68% of the original number. The running time is reduced from 1511.70 to 426.79 s, *i.e.*, to 28.17% of the original time.

3. BIASED RANDOM-KEY GENETIC ALGORITHMS FOR MAXIMUM QUASI-CLIQUE

In a biased random-key genetic algorithm (BRKGA), each solution is represented by a vector of randomly generated real numbers called keys. A deterministic algorithm, called a decoder, takes as input a solution represented by a vector of random keys and associates with it a feasible solution of the combinatorial optimization problem at hand, for which an objective value or fitness can be computed. Selection in a BRKGA is said to be biased because one of the two parents selected for mating in each crossover operation is always an elite solution and has a higher probability of passing its genes to the new generation, while the other is a non-elite solution. The reader is referred to Gonçalves and Resende [11] and Resende and Ribeiro [24] for complete reviews about biased random-key genetic algorithms and their applications.

Two variants of a BRKGA for MQCP have been developed in [23], each of them using a different decoder. The algorithms evolve a population that is formed by vectors of real-valued components in the range [0, 1), each component being associated with one of the vertices of the graph G. Each vector is decoded by a deterministic





FIGURE 1. Population evolution between consecutive generations of a BRKGA [23].

algorithm that builds a feasible solution for MQCP, *i.e.*, a γ -clique. The two decoders DECODER-HCB and DECODER-IG^{*} originally presented in [23] are briefly summarized in Section 4.

The parametric uniform crossover of Spears and de Jong [29] is used to combine two mates: the offspring inherits with higher probability each of its keys from the best fit of the two parents. Instead of the standard mutation operator, the concept of mutants is used: randomly generated mutant solutions are introduced in the population in each generation. Mutants play the same role of the mutation operator in traditional genetic algorithms, diversifying the search and helping the procedure to escape from locally optimal solutions (see also [6, 7, 17]).

The initial population is randomly generated. The population is partitioned into two subsets at each generation: TOP and REST. Subset TOP contains the elite solutions, being formed by the best solutions in the population of the current generation. Subset REST is decomposed in two subsets: MID and BOT, with subset BOT being formed by the worst elements in the population of the current generation. The size of the population is |TOP| + |REST|.

The population evolves from one generation to the next as illustrated in Figure 1. First, the solutions in TOP are simply copied to the population of the next generation. The elements in BOT are replaced by new randomly created mutants that are placed in the new set BOT. The remaining |MID| = |REST| - |BOT| solutions of the new population are obtained by crossover, with one parent randomly chosen from TOP and the other from REST. This is the main difference between a BRKGA and the random-key genetic algorithm of Bean [4], where both parents are randomly selected from the entire population: since a solution can be chosen for mating more than once in any given generation, elite solutions have a higher probability of passing their random keys to the next generation.

The variants of the BRKGA for the maximum quasi-clique problem were implemented with the C++ library brkgaAPI developed by Toso and Resende [30], which is a framework illustrated in Figure 2 for the development of biased random-key genetic algorithms. Its instantiation to some specific optimization problem requires only the development of a class implementing the decoder for the problem at hand, since this is the only problem-dependent part of the tool.

The brkgaAPI framework requires the following parameters [12]: (a) the population size p = |TOP| + |REST|; (b) the fraction *pe* of the population corresponding to the elite subset TOP; (c) the fraction *pm* of the population corresponding to the mutant subset BOT; (d) the probability *rhoe* that the offspring inherits the keys from the best fit parent; and (e) the number k of generations without improvement in the best solution until a restart is performed.


FIGURE 2. BRKGA framework [23].

In the remainder of this work, we consider and compare two variants of a biased random-key genetic algorithm for solving MQCP, based on two different decoders. Decoder DECODER-IG^{*} was originally proposed by Pinto *et al.* [23] and will be summarized in the next section. The new decoder, named DECODER-LSQClique and proposed in this work, is based on the exact enumeration algorithm of Ribeiro and Riveaux [25] and will be presented in Section 5.

4. Decoder DECODER-IG*

Decoder DECODER-HCB was firstly presented in [23] and finds its roots in the construction phase of the GRASP approach of Abello *et al.* [1]. It takes as inputs the graph G = [V, E], the threshold γ , the parameter α used to create the restricted candidate lists, the parameter *minsize* that defines the minimum size of the restricted candidate lists, and the subset S of vertices in the initial solution. The decoder also receives as parameters the random keys $R_j \in [0, 1), j = 1, \ldots, |V|$, each of them corresponding to a vertex of the graph. Each vector of random keys is decoded by an algorithm that builds a feasible solution to MQCP, *i.e.*, a γ -clique. The pseudo-code of this decoder appears in Algorithm 1.

Algorithm 1 may be used in two situations. First, to build a solution from scratch. Second, to complete (i.e., to reconstruct) a partially destroyed solution. For the second case, the decoder receives as an additional parameter a partial solution formed by a non-empty list S of vertices.

The second decoder, named DECODER-IG^{*}, was also originally presented in [23] and is an extension of DECODER-HCB. It follows the iterated greedy scheme, using the constructive heuristic HCB. The population is formed by longer vectors of $2 \cdot |V|$ random keys $R_j \in [0, 1)$ each, with $j = 1, \ldots, |V|, |V| + 1, \ldots, 2 \cdot |V|$. The first |V| random keys are used in the construction of the initial solution and in the reconstruction phase, while the last |V| random keys are used in the destruction phase. Algorithm 2 takes two additional parameters as inputs: δ and β . Parameter δ controls the fraction of the vertices of the current solution that will be removed, while parameter β determines the greediness of the removal process, by controlling the size of the restricted candidate list from where each vertex will be extracted. The role played by each parameter is explained with more detail in [23]. The pseudo-code of this decoder appears in Algorithm 2.

The algorithm starts in line 1 building the initial solution S' with decoder DECODER-HCB and the first |V| random keys $R_j, j = 1, ..., |V|$. Lines 2 to 10 implement the loop that repeats the partial destruction (vertex eliminations) followed by the reconstruction (vertex insertions) of the current solution, until no further

Algorithm 1. DECODER-HCB $(G, \gamma, \alpha, minsize, S, R)$.

```
1: CL \leftarrow V \setminus S
 2: if S = \emptyset then
 3:
           RCL \leftarrow \{v \in CL : |\{v' \in CL : deg_G(v') \ge deg_G(v)\}| \le \max\{minsize, \lfloor \alpha \cdot |CL| \rfloor\}\}
 4:
           x \leftarrow argmin\{R_j : j \in RCL\}
           S \leftarrow \{x\}
 5:
 6: end if
 7: while CL \neq \emptyset do
           CL \leftarrow \emptyset
 8:
           for all v \in V \setminus S do
 9:
                if \frac{2 \cdot (|E(S)| + deg_G(v,S))}{|S|(|S|+1)} \ge \gamma then
10:
                     CL \leftarrow CL \cup \{v\}
11:
                end if
12:
13:
           end for
           if CL \neq \emptyset then
14:
15:
                for all v \in CL do
16:
                     dif(v) \leftarrow \deg_G(v, CL) + |CL| \cdot (\deg_G(v, S) - \gamma \cdot (|S| + 1))
                end for
17:
                RCL \leftarrow \{v \in CL : |\{v' \in CL : dif(v') \ge dif(v)\}| \le \max\{minsize, |\alpha \cdot |CL|\}\}
18:
                x \leftarrow argmin\{R_j : j \in RCL\}
19:
20:
                S \leftarrow S \cup \{x\}
          end if
21:
22: end while
```

improvements can be obtained. The current solution S' is copied to S in line 3. Lines 4 to 8 implement the loop that removes one by one $\lfloor \delta \cdot |S'| \rfloor$ vertices that should be eliminated from the current solution. In each step, in line 5 a restricted candidate list RCL of size max{minsize, $\lfloor \beta \cdot |S'| \rfloor$ } is created containing the vertices with the smallest degrees in G(S'). The vertex with the smallest random key is selected from RCL in line 6 and eliminated from the current solution in line 7. Line 9 performs the reconstruction phase: the current partial solution S' is rebuilt by decoder DECODER-HCB using the first |V| random keys. The loop is interrupted in line 10 when the new solution S' obtained by destruction-reconstruction does not improve the incumbent S or the graph G(S') is not connected; otherwise a new iteration resumes.

Algorithm 2. DECODER-IG^{*}($G, \gamma, \alpha, \delta, \beta, minsize, R$). 1: $S' \leftarrow \text{DECODER-HCB}(G, \gamma, \alpha, minsize, \emptyset, R_j : j = 1, \dots, |V|)$ 2: repeat 3: $S \leftarrow S'$ 4: for k = 1 to $\lfloor \delta \cdot |S'| \rfloor$ do $RCL \leftarrow \{v \in S' : |\{v' \in S' : deg_G(v', S') \le deg_G(v, S')\}| \le \max\{minsize, \lfloor\beta \cdot |S'|\rfloor\}\}$ 5: $x \leftarrow argmin\{R_{|V|+j} : j \in RCL\}$ 6: $S' \leftarrow S' \setminus \{x\}$ 7: 8: end for $S' \leftarrow \text{DECODER-HCB}(G, \gamma, \alpha, \textit{minsize}, S', R_j : j = 1, \dots, |V|)$ 9: 10: **until** $|S'| \leq |S|$ or graph G(S') is not connected

5. New decoder based on exact local search

The principle of the new decoder proposed in this section consists in replacing the reconstruction phase of decoder DECODER-IG^{*} (*i.e.*, line 9 of Algorithm 2) by an exact algorithm that finds the maximum quasi-

S746

clique that can be reconstructed by DECODER-HCB starting from the remaining set of vertices S' with $\alpha = 1$. A slightly modified version of the *QClique* algorithm of C.C. Ribeiro and J.A. Riveaux [25] is used with this goal.

Algorithm 3 describes the pseudo-code of decoder DECODER-LSQClique. The algorithm takes as inputs the same parameters as DECODER-IG^{*}, with the exception of an extended vector with $2 \cdot |V| + 1$ random keys $R_j^+, j = 1, \ldots, 2 \cdot |V|, 2 \cdot |V| + 1$ and one additional frequency parameter ρ : the exact search algorithm will be applied whenever $R_{2 \cdot |V|+1}^+ \leq \rho$, otherwise decoder DECODER-HCB will be used to reconstruct the solution. We note that, if $\rho = 1$ the exact search algorithm will always be executed. If $\rho = 0$, it will never be executed and DECODER-LSQClique will behave exactly as DECODER-IG^{*}. We observe that by appropriately tuning and setting the best value for ρ one can optimize the performance of this decoder.

DECODER-LSQClique starts by creating an initial solution S' in line 1, using DECODER-HCB and the random keys $R_j, j = 1, \ldots, |V|$. The loop in lines 2 to 14 repeats the partial destruction (vertex eliminations) followed by the reconstruction (vertex insertions) of the current solution, until no further improvements can be obtained. The current solution S' is copied to S in line 3. The loop in lines 4 to 8 removes one by one the $\lfloor \delta \cdot |S'| \rfloor$ vertices that should be eliminated from the current solution. A restricted candidate RCL of size max $\{minsize, \lfloor \beta \cdot |CL| \rfloor\}$ is created in line 5, containing the vertices with the smallest degrees in G(S'). The vertex with the smallest random key $R_{|V|+j}, j \in RCL$, is selected from the restricted candidate list in line 6 and eliminated from the current solution in line 7. The reconstruction phase starts in line 9. If the random key $R_{2:|V|+1}^+$ is smaller than or equal to parameter ρ and G(S') is a γ -clique, then the partial solution S' is extended in line 10 by an exact local search algorithm and a new solution S^* is obtained. Otherwise, solution S' is rebuilt by DECODER-HCB in line 12, once again using the first random keys $R_j^+, j = 1, \ldots, |V|$. The loop is interrupted in line 14 when the new solution S' obtained by destruction-reconstruction does not improve the incumbent S or the graph G(S') is not connected; otherwise a new iteration resumes. The best solution is returned in S.

Algorithm 3. DECODER-LSQClique($G, \gamma, \alpha, \delta, \beta, minsize, S, R^+, \rho$).

1: $S' \leftarrow \text{DECODER-HCB}(G, \gamma, \alpha, minsize, \emptyset, R_j^+, j = 1, \dots, |V|)$ 2: repeat $S \leftarrow S'$ 3: 4: for k = 1 to $|\delta \cdot |S'||$ do $RCL \leftarrow \{v \in S' : |\{v' \in S' : deg_G(v', S') \leq deg_G(v, S')\}| \leq \max\{minsize, \lfloor \beta \cdot |S' \rfloor \}\}$ 5: 6: $x \leftarrow argmin\{R_{|V|+j} : j \in RCL\}$ $S' \leftarrow \bar{S'} \setminus \{x\}$ 7: end for 8: 9: if $R_{2 \cdot |V|+1}^+ \leq \rho$ and $dens(G(S')) \geq \gamma$ then $LSQClique(G, S', \gamma, |S|, S^*)$ 10:11: else $S' \leftarrow \text{DECODER-HCB}(G, \gamma, \alpha, minsize, S', R_i^+ : j = 1, \dots, |V|)$ 12:13:end if 14: **until** $|S'| \leq |S|$ or graph G(S') is not connected

6. Exact algorithm for New Decoder

Algorithm *LSQClique* described in this section is an exact algorithm for the maximum quasi-clique problem. This variant may investigate multiple permutations of the same solution along the search tree, because it does not make use of the quasi-heredity proposition [25]. It will be used as a variant of algorithm *QClique* in line 10 of Algorithm 3 to perform the exact local search.

B.Q. PINTO ET AL.

Algorithm 4.	LSQClique	(G,	$S', \gamma,$	LB,	S^*).
--------------	-----------	-----	---------------	-----	-------	----

1: $S^* \leftarrow S'$ $2:\ i \leftarrow |S'|$ 3: if a solution of size k containing all vertices of S' does not exist for some $k = i + 1, \ldots, LB + 1$ then return 4: $CL \leftarrow \{v \in V \setminus S' : dens(G(S' \cup \{v\})) \ge \gamma\}$ 5: if $CL = \emptyset$ then return 6: for all $j \in CL$ do $LSQClique(G, S' \cup \{j\}, \gamma, \max\{LB, |S'| + 1\}, S'')$ 7: $\mathbf{if}\ |S''| > |S^*| \ \mathbf{then}$ 8: $S^* \leftarrow S''$ 9: 10: $LB \leftarrow \max\{LB, |S^*|\}$ end if 11: 12: end for

Algorithm 4 works directly with the initial solution represented by the set of vertices S'. It takes as inputs the graph G, the threshold γ , the partial solution S', and the initial lower bound LB = |S|, returning the best solution found S^* also as a parameter. Line 1 of the pseudo-code copies to S^* the current partial solution S'. Line 2 sets i with the size of the current solution S'. If a solution of size k containing all vertices in S' does not exist for some value of $k = i + 1, \ldots, LB + 1$, then the algorithm returns in line 3 (pruning).

The candidate list CL created in line 4 is formed by every vertex $v \in V \setminus S' : dens(G(S' \cup \{v\})) \ge \gamma$, as for DECODER-HCB. If the candidate list is empty, then the algorithm returns in line 5.

The loop in lines 6 to 12 attempts to extend the current solution S' by each vertex j in the candidate list CL. The vertices in the candidate list are taken in non increasing order of their degrees $deg_G(j, S')$. For each of them, line 7 recursively invokes the algorithm for a new, extended partial solution $S' \cup \{j\}$ and a new, updated lower bound max $\{LB, |S'| + 1\}$, returning a new solution S''. Line 8 checks if the new solution S'' improves then current best S^* . If this is the case, the current best and the lower bound are updated in lines 9 and 10, respectively. When the recursion terminates, S^* contains the best solution obtained by the algorithm.

The pruning strategy discards the current node of the enumeration tree in line 3 if a solution S of size k containing all vertices of S' can not exist for some k = i + 1, ..., LB + 1. This is done by calculating an upper bound to the number of edges in G(S) and checking if the number of edges in a γ -clique is greater than this upper bound. We follow a similar idea to that in [25].

Let $E(S', S \setminus S')$ denote the set of edges in G between the vertices of S' and $S \setminus S'$. If S is a γ -clique, then the following condition holds and gives a lower bound to |E(S)|:

$$|E(S)| = |E(S') \cup E(S', S \setminus S') \cup E(S \setminus S')| \ge \gamma \cdot \binom{k}{2}.$$
(6.1)

Since the sets in equation (6.1) are disjoint,

$$|E(S)| = |E(S')| + |E(S', S \setminus S')| + |E(S \setminus S')|.$$
(6.2)

Since |E(S')| is already known, in order to calculate an upper bound to |E(S)| we need to calculate an upper bound to $|E(S', S \setminus S')| + |E(S \setminus S')|$.

Let V' be formed by the k - i vertices of $V \setminus S'$ with the maximum number of adjacent vertices in G that belong to S'. V' can be determined in polynomial time O(|V|): first sort the vertices in the non increasing order of their degrees $deg_G(v, S)$, then select the k - i first vertices. Then, an upper bound to $|E(S', S \setminus S')|$ is given by |E(S', V')|.

Now, let us define V'' as the set formed by the k - i vertices of $V \setminus S'$ with maximum degree in G. Then, the following condition holds:

v

$$\sum_{\in S \setminus S'} \deg_G(v) \le \sum_{v \in V''} \deg_G(v).$$
(6.3)

S748

Following a reasoning similar to that in [25], we obtain:

$$|E(S', S \setminus S')| + |E(S \setminus S')|$$

$$\leq \sum_{v \in V'} \deg_G(v, S') + \min\left\{\frac{\sum_{v \in V''} \deg_G(v) - \sum_{v \in V'} \deg_G(v, S')}{2}, \binom{k-i}{2}\right\}.$$
(6.4)

Therefore, if the inequality below is not satisfied for at least one value of k = i + 1, ..., LB + 1, then a solution of size LB + 1 does not exist and we can prune the current node of the search tree:

$$|E(S')| + \sum_{v \in V'} \deg_G(v, S') + \min\left\{\frac{\sum_{v \in V''} \deg_G(v) - \sum_{v \in V'} \deg_G(v, S')}{2}, \binom{k-i}{2}\right\}$$

$$\geq \gamma \cdot \binom{k}{2}.$$
(6.5)

7. Computational results

7.1. Experimental setting

All algorithms were implemented using version 19.14.26429.4 of the Microsoft[®] C/C++ Optimizing Compiler for x64. The computational experiments have been performed on an Intel Core i5-5200 processor with 2.20 GHz and 8 GB of RAM running under Windows 10.

The newly proposed algorithm BRKGA-LSQClique was compared with the original BRKGA-IG^{*} heuristic of [23], which was the best heuristic for the maximum quasi-clique problem at the time of writing. The best parameters for algorithm BRKGA-IG^{*} were determined using the automatic tuning tool IRACE [16, 22]. The same parameter settings were used for algorithm BRKGA-LSQClique, except for the parameter δ that controls the number of vertices to be destroyed. In order to enforce that the fraction of nodes eliminated in the destruction phase be smaller for dense graphs, DECODER-LSQClique empirically sets $\delta = 1 - dens(G)$, if dens(G) < 0.8; $\delta = 2 \cdot (1 - dens(G))$, otherwise.

DECODER-LSQClique makes use of an additional parameter ρ to establish whether the exact local search should be applied or not to some solution. Again, we empirically set the probability that exact local search is applied at $\rho = \max\{1 - dens(G), 0.80\}$.

Implementation notice. An additional parameter maxnodes was used to enforce a maximum limit to the number of nodes of the search tree generated by the exact algorithm LSQClique. In case the search tree generated by LSQClique reaches this maximum limit of nodes, then the algorithm returns the incumbent before an optimal solution is obtained. For this reason, an additional application of the reconstruction procedure

$$S' \leftarrow \text{DECODER-HCB}(G, \gamma, \alpha, minsize, S^*, R_j^+ : j = 1, \dots, |V|)$$

should be enforced after line 10 of Algorithm 3 to further improve the current solution. This parameter *maxnodes* was set at 220 using the IRACE tool.

All tested value ranges and the best parameter settings are shown in Table 1.

7.2. Numerical results

Pinto *et al.* [23] presented numerical results for experiments on 67 DIMACS instances [14, 28], 33 maximum clique instances of the Benchmarks with Hidden Optimum Solutions for Graph Problems (BHOSLIB) [5, 26, 27], and six sparse instances from the University of Florida Sparse Matrix Collection [9]. In the experiments reported in the section, we eliminated all instances with less than 400 vertices or with density greater than or equal to 0.99, therefore excluding 29 instances from [14, 28] and two instances from [9] that were used in [23]. We also considered three large sparse instances from [9] that were used by Veremyev *et al.* [31] (Geom, EVA and

Parameter	Value ranges	BRKGA-IG*	BRKGA-LSQClique
p	$50, 51, \ldots, 100$	91	91
p_e	$0.10, 0.11, \ldots, 0.25$	0.13	0.13
p_m	$0.10, 0.11, \ldots, 0.30$	0.22	0.22
rhoe	$0.50, 0.51, \ldots, 0.80$	0.78	0.78
α	$0.01, 0.02, \ldots, 0.20$	0.01	0.01
minsize	1, 2, 3, 4, 5, 6	3	3
δ	$0.01, 0.02, \ldots, 0.50$	0.40	$\begin{cases} 1 - dens(G); \text{ if } dens(G) < 0.8\\ 2 \cdot (1 - dens(G)); \text{ otherwise} \end{cases}$
β	$0.01, 0.02, \ldots, 0.20$	0.02	0.02
maxnodes	$100, 110, \ldots, 300$	—	220
ρ	_	_	$\max\{1 - dens(G), 0.80\}$

TABLE 1. Parameters settings: parameter maxnodes was set using IRACE in this work.

Notes. Parameters ρ and δ were empirically set in this work. All other parameters were set using IRACE by Pinto *et al.* [23].

PGPgiantcompo) and instance vsp_p0291_seymourl_iiasa used by Ribeiro and Riveaux [25]. Table 2 describes the characteristics of each instance.

In the first experiment, for each instance and each value of the threshold γ , we performed 30 runs of each algorithm BRKGA-IG^{*} and BRKGA-LSQClique until a target solution value was found. For each instance and value of γ , the target was determined as follows. We run both BRKGA-IG^{*} and BRKGA-LSQClique for 100 generations or ten minutes, whatever happened first, and selected the target as the best among the solution values found by the two algorithms. For some instances for which this value revealed itself to be easy, we progressively increased the target until a sufficiently hard value was reached.

We discarded the results for the combinations of instance and value of γ for which both algorithm obtained the target solution value in less than two generations on average over the 30 runs. Therefore, Tables 3–9 display the results for 150 combinations of different values of γ with 24 of the 38 DIMACS instances, with all 33 BHOSLIB instances, and with four of the eight sparse instances that were not completely discarded. Each run was limited to ten minutes of execution, except for the instances in Table 9, whose runs were limited to 30 minutes of execution. For each instance and value of γ , the tables show the target look4 considered in the experiment, and the results obtained by each algorithm BRKGA-IG^{*} and BRKGA-LSQClique. For each algorithm, the table presents the average generation in which the target was found over the 30 runs, the average running time in seconds and the average generation in which the target was found over the runs that reached the target value (#look4). Whenever any of the two methods failed to find a solution as good as the target value look4 within the time limit for all instances in the group of same density, we indicate it by "> 600 (0)" or "> 1800 (0)".

The rows highlighted in dark gray () correspond to the cases where BRKGA-IG^{*} reached the target **look4** more often or, when both of them reached the target the same number of times, it was faster than BRKGA-LSQClique in terms of the average time to target. Rows in light gray () are those for which BRKGA-LSQClique performed better in terms of the same criteria. For all other cases, both algorithms found the target in the first generation in all runs.

For each instance and each value of γ in Tables 3–9, the two algorithms are compared primarily on the number of runs in which the target solution value was found and, whenever there is a tie, on the average time taken to find the target solution value over the successful runs. The plot in Figure 3 displays the comparison between algorithms BRKGA-IG^{*} and BRKGA-LSQClique. For each value of γ , it indicates the fraction of the number of instances for which each algorithm outperforms the other. Considering all 150 combinations of instances and values of γ , BRKGA-LSQClique outperformed the original BRKGA-IG^{*} in 63.33% of the cases in terms of the time-to-target solution value.

Instance	V	E	Density	Instance	V	E	Density
$san400_0.5_1$	400	39900	0.50	frb45-21-1	945	386854	0.87
$sanr400_0.5$	400	39984	0.50	frb45-21-2	945	387416	0.87
$san400_0.7_1$	400	55860	0.70	frb45-21-5	945	387461	0.87
$san400_0.7_2$	400	55860	0.70	frb45-21-4	945	387491	0.87
$san400_0.7_3$	400	55860	0.70	p_hat1000-1	1000	122253	0.24
$sanr400_0.7$	400	55869	0.70	p_hat1000-2	1000	244799	0.49
brock400_3	400	59681	0.75	DSJC1000.5	1000	249826	0.50
brock400_1	400	59723	0.75	san1000	1000	250500	0.50
brock400_2	400	59786	0.75	p_hat1000-3	1000	371746	0.74
gen400_p0.9_55	400	71820	0.90	C1000.9	1000	450079	0.90
gen400_p0.9_65	400	71820	0.90	hamming10-4	1024	434176	0.83
frb30-15-2	450	83151	0.82	email	1133	5451	0.01
frb30-15-4	450	83194	0.82	frb50-23-2	1150	579824	0.88
frb30-15-1	450	83198	0.82	frb50-23-4	1150	580417	0.88
frb30-15-5	450	83231	0.82	frb50-23-1	1150	580603	0.88
Erdos971	472	1314	0.01	frb50-23-5	1150	580640	0.88
johnson32-2-4	496	107880	0.88	frb53-24-4	1272	714048	0.88
Harvard500	500	2043	0.02	frb53-24-2	1272	714067	0.88
c-fat500-1	500	4459	0.04	frb53-24-1	1272	714129	0.88
c-fat500-2	500	9139	0.07	frb53-24-5	1272	714130	0.88
c-fat500-5	500	23191	0.19	frb56-25-4	1400	869262	0.89
p_hat500-1	500	31569	0.25	frb56-25-1	1400	869624	0.89
c-fat500-10	500	46627	0.37	frb56-25-5	1400	869699	0.89
DSJC500.5	500	62624	0.50	frb56-25-2	1400	869899	0.89
p_hat500-2	500	62946	0.50	p_hat1500-2	1500	568960	0.51
p_hat500-3	500	93800	0.75	frb59-26-4	1534	1048800	0.89
C500.9	500	112332	0.90	frb59-26-1	1534	1049256	0.89
frb35-17-5	595	148572	0.84	frb59-26-2	1534	1049648	0.89
frb35-17-1	595	148859	0.84	frb59-26-5	1534	1049829	0.89
frb35-17-2	595	148868	0.84	C2000.9	2000	1799532	0.90
frb35-17-4	595	148873	0.84	keller6	3361	4619898	0.82
p_hat700-1	700	60999	0.25	C4000.5	4000	4000268	0.50
p_hat700-2	700	121728	0.50	frb100-40	4000	7425226	0.93
frb40-19-5	760	246801	0.86	CA-GrQc	5242	14496	$1.1/10^{3}$
frb40-19-4	760	246815	0.86	Geom	7343	11898	$1.9/10^{4}$
frb40-19-1	760	247106	0.86	EVA	8497	6711	$3.4/10^4$
frb40-19-2	760	247157	0.86	vsp_p0291-	10498	53868	$9.8/10^4$
keller5	776	225990	0.75	_seymourl_iiasa			
brock800_3	800	207333	0.65	PGPgiant-	10680	24316	$4.3/10^{4}$
brock800_1	800	207505	0.65	compo			
brock800_2	800	208166	0.65				

TABLE 2. Description of the test instances.

In the next experiment, we evaluate and compare the run time distributions (or time-to-target plots) of algorithms BRKGA-IG^{*} and BRKGA-LSQClique for some instances. Time-to-target plots display on the ordinate axis the probability that an algorithm will find a solution at least as good as a given target value within a given running time, shown on the abscissa axis. Run time distributions have also been advocated by Hoos and Stützle [13] as a way to characterize the running times of stochastic local search algorithms for combinatorial optimization problems. In this experiment, the two algorithms were made to stop whenever a solution with cost greater than or equal to a given target value was found. The targets are the same used in the first experiment.

	Avg. gen.	25.97	1.70	2.47	920.95	3.87	5.00	3.40	9.10	20.93	2.47	1504.56	14.37	1.27	22.43	3.23	1.63	1498.67	149.10	119.60	11.60	196.72	332.63	184.75	150.90
KGA-LSQClique	Time (#100k4)	6.67(30)	1.46(30)	0.54(30)	245.54(20)	0.87(30)	3.60(30)	3.31(30)	1.57(30)	14.58(30)	6.09(30)	294.50(25)	10.78(30)	1.84(30)	4.81(30)	2.30(30)	1.74(30)	212.16(3)	26.00(30)	24.96(30)	7.96(30)	221.01(25)	75.50(30)	170.38(24)	172.72(29)
BR	Best	13	32	40	22	21	46	113	25	61	187	25	60	189	25	61	186	54	66	30	58	138	29	61	138
	Avg.	13.00	32.00	40.00	20.30	21.00	46.00	113.00	25.00	61.00	187.00	24.83	60.00	189.00	25.00	61.00	186.00	53.10	66.00	29.03	58.00	137.83	29.00	60.80	137.97
	Avg. gen.	49.33	3.20	1.77	1153.10	3.03	9.43	7.50	10.47	56.03	39.60	852.43	47.30	3.10	13.10	3.67	6.00	2554.47	64.47	84.63	19.40	318.17	394.97	364.58	303.10
BRKGA-IG*	Time (#100k4)	6.21(30)	1.99(30)	0.26(30)	141.01(29)	0.39(30)	4.88(30)	5.24(30)	1.25(30)	29.52(30)	29.11(30)	101.51(30)	23.60(30)	2.62(30)	1.57(30)	1.88(30)	4.50(30)	340.34(15)	8.73(30)	13.16(30)	11.38(30)	276.64(12)	58.10(30)	243.11(24)	255.28(21)
	Best	13	32	40	22	21	46	113	25	61	187	25	60	189	25	61	186	54	66	30	58	138	29	61	138
	Avg.	13.00	32.00	40.00	21.30	21.00	46.00	113.00	25.00	61.00	187.00	25.00	60.00	189.00	25.00	61.00	186.00	53.50	66.00	29.13	58.00	137.40	29.00	60.80	137.70
	look4	13	32	40	20	21	46	113	25	61	187	25	60	189	25	61	186	54	66	29	58	138	29	61	138
	Х	0.999	0.80	0.999	0.999	0.999	0.90	0.80	0.999	0.90	0.80	0.999	0.90	0.80	0.999	0.90	0.80	0.999	0.999	0.999	0.95	0.90	0.999	0.95	0.90
	Instance	$\operatorname{sanr400_{-0.5}}$		$san400_{-}0.7_{-}1$	$san400_{-0.7_{-3}}$	$\operatorname{sanr4000.7}$			brock400_3			brock400_1			brock400_2			gen400_p0.9_55	gen400_p0.9_65	frb30-15-2			frb30-15-4		

TABLE 3. First experiment: 30 runs of each instance and algorithm limited to 600s of running time each.

B.Q. PINTO ET AL.

		Avg. gen.	171.73	9.80	11.37	694.60	47.63	11.57	3.10	10.13	23.63	55.10	4.83	87.80	23.07	71.33	372.70	47.77	2.30	115.53	23.03	23.40	247.93	9.27	98.00	14.77	1.27
)	KGA-LSQClique	Time (#100k4)	34.37(30)	7.20(30)	11.57(30)	141.80(30)	41.69(30)	11.39(30)	2.89(30)	10.85(30)	8.14(30)	11.12(30)	4.01(30)	24.78(30)	33.77(30)	109.48(30)	108.88(30)	67.17(30)	4.31(30)	32.79(30)	33.02(30)	43.63(30)	77.14(30)	13.27(30)	155.42(30)	8.93(30)	1.56(30)
	BR	Best	29	59	139	29	00	134	21	34	51	57	157	33	80	225	33	78	216	33	76	207	33	81	238	11	22
		Avg.	29.00	59.00	139.00	29.00	60.00	134.00	21.00	34.00	51.00	57.00	157.00	33.00	79.03	225.00	33.00	78.00	216.00	33.00	75.10	207.00	33.00	79.17	238.00	11.00	22.00
)		Avg. gen.	126.47	13.07	31.47	375.47	157.43	24.60	4.87	21.70	22.57	49.40	10.17	60.90	69.50	252.63	168.80	92.87	3.97	85.87	63.47	60.60	187.57	18.93	170.00	23.13	2.07
	BRKGA-IG*	Time (#100k4)	18.90(30)	7.31(30)	25.30(30)	55.16(30)	99.01(30)	19.61(30)	3.04(30)	20.66(30)	5.07(30)	10.79(30)	8.72(30)	15.21(30)	86.15(30)	389.49(16)	41.66(30)	112.96(29)	6.75(30)	21.08(30)	81.34(30)	100.84(30)	47.44(30)	23.49(30)	258.20(23)	4.10(30)	1.81(30)
		Best	29	59	139	29	00	134	21	34	51	57	157	34	62	225	33	78	216	33	26	207	33	80	238	11	22
		Avg.	29.00	59.00	139.00	29.00	60.00	134.00	21.00	34.00	51.00	57.00	157.00	33.03	79.00	224.53	33.00	77.97	216.00	33.00	75.10	207.00	33.00	79.07	237.77	11.00	22.00
		look4	29	59	139	29	60	134	21	34	51	57	157	33	62	225	33	78	216	33	75	207	33	62	238	11	22
		Х	0.999	0.95	0.90	0.999	0.95	0.90	0.90	0.80	0.999	0.999	0.95	0.999	0.95	0.90	0.999	0.95	0.90	0.999	0.95	0.90	0.999	0.95	0.90	0.999	0.80
		Instance	frb30-15-1			frb30-15-5			DSJC500.5		p_hat500-3	C500.9		frb35-17-5			frb35-17-1			frb35-17-2			frb35-17-4			p_hat700-1	

TABLE 4. First experiment: 30 runs of each instance and algorithm limited to 600s of running time each.

	Avg. gen.	282.14	51.97	13.17	573.72	9.33	90.50	73.73	136.91	55.23	30.57	2.83	45.77	24.17	12.30	20.27	135.43	26.67	19.03	189.13	39.59	11.27	37.90	3.63	2.20
KGA-LSQClique	Time (#100k4)	120.16(29)	123.23(29)	32.61(30)	252.25(18)	22.56(30)	229.11(22)	32.11(30)	340.15(11)	23.98(30)	71.23(30)	6.77(30)	111.25(30)	15.97(30)	28.47(30)	59.67(30)	91.67(30)	68.18(30)	58.27(30)	131.81(30)	98.38(29)	33.99(30)	26.03(30)	15.65(30)	7.10(30)
BR	Best	38	100	333	38	96	292	38	114	37	104	363	132	21	42	93	21	42	96	21	43	94	42	124	508
	Avg.	37.97	99.97	333.00	37.60	95.20	291.73	37.03	113.30	37.00	103.07	363.00	132.00	21.00	42.00	93.00	21.00	42.00	96.00	21.00	42.00	94.00	41.03	121.57	508.00
	Avg. gen.	254.43	115.07	87.38	370.67	24.57	160.50	68.90	232.00	40.13	82.04	6.80	41.03	18.67	79.76	78.91	108.77	54.88	80.22	106.90	70.07	30.87	18.97	10.37	10.27
BRKGA-IG*	Time (#look4)	99.55(30)	264.97(15)	219.13(24)	145.57(24)	53.74(30)	395.01(4)	27.10(30)	504.90(2)	15.78(30)	178.76(26)	16.89(30)	92.79(30)	11.83(30)	190.77(29)	237.31(22)	50.07(30)	143.39(26)	254.22(23)	54.91(30)	182.78(28)	94.59(30)	12.20(30)	43.20(30)	35.54(30)
	Best	38	100	333	39	95	292	37	114	37	104	363	132	21	42	93	21	42	96	21	42	95	42	123	508
	Avg.	38.00	99.50	332.80	37.83	95.00	291.10	37.00	112.63	37.00	102.90	363.00	132.00	21.00	41.97	92.73	21.00	41.87	95.77	21.00	41.93	94.03	41.07	121.33	508.00
	look4	38	100	333	38	95	292	37	114	37	103	363	132	21	42	93	21	42	96	21	42	94	41	121	508
	λ	0.999	0.95	0.90	0.999	0.95	0.90	0.999	0.95	0.999	0.95	0.90	0.90	0.999	0.90	0.80	0.999	0.90	0.80	0.999	0.90	0.80	0.999	0.95	0.90
	Instance	frb40-19-5			frb40-19-4			frb40-19-1		frb40-19-2			keller5	brock800_3			brock800_1			brock800_2			frb45-21-1		

TABLE 5. First experiment: 30 runs of each instance and algorithm limited to 600s of running time each.

B.Q. PINTO ET AL.

BRKGA-IG* look4 Avg. Best Time (#look4) Avg. gen
41 41.07 42 15.26(30) 23.
120 120.23 122 71.53(30) 1
491 490.17 491 291.03(5) 7
42 41.90 43 $195.14(26)$ 31
121 120.93 123 177.89(27)
$508 \ 507.13 \ 508 \ 176.41(4)$
42 42.00 42 125.88(30) 1
$128 \ 1280 \ 128 \ 12800 \ 128 \ 126.04(30)$
554 554.00 554 17.76(30)
15 15.00 15 $2.48(30)$
23 23.00 23 $48.94(30)$
15 14.67 15 265.43(20)
24 23.30 24 $254.91(9)$
39 38.87 40 242.46(27)
67 67.10 68 126.51(30)
217 217.30 219 $128.51(30)$
40 40.00 40 30.99(30)
83 82.10 83 249.13(3)
46 46.03 47 27.28(30)
$158 \ 156.13 \ 158 \ 397.67(6)$
780 779.93 780 154.18(28)
47 47.00 47 $159.94(30)$
$157 \ 156.43 \ 158 \ 350.15(19)$
772 771.00 771 $0.00(0)$

TABLE 6. First experiment: 30 runs of each instance and algorithm limited to 600s of running time each.

	Avg. gen.	38.87	6.63	8.13	29.43	26.43	1.17	107.75	5.80	77.08	18.00	13.00	0.00	10.93	2.63	65.26	5.50	11.70	37.00	7.47	56.00	3.27	1.00	45.50	11.41
KGA-LSQClique	Time (#100k4)	70.75(30)	54.86(30)	52.13(30)	54.29(30)	201.76(30)	8.17(30)	317.79(4)	78.62(30)	216.89(12)	202.27(29)	123.69(27)	>600(0)	142.60(30)	23.79(30)	194.90(23)	69.86(30)	125.81(30)	131.59(29)	128.24(30)	188.83(1)	69.21(30)	11.05(30)	165.91(8)	188.53(30)
BRI	Best	47	160	797	47	162	788	50	180	50	174	926	49	201	978	50	171	894	52	197	53	227	1161	52	203
	Avg.	46.03	157.53	797.00	46.13	160.43	788.00	48.70	178.40	49.30	172.07	925.90	48.77	199.50	978.00	48.80	167.13	894.00	51.00	195.33	51.20	225.33	1161.00	51.23	201.50
	Avg. gen.	15.97	28.46	11.00	13.60	33.50	10.00	91.40	19.18	64.38	26.20	25.00	123.83	20.88	27.46	54.83	20.58	I	25.20	10.00	66.00	10.80	4.27	66.40	23.33
3RKGA-IG*	Time (#100k4)	29.85(30)	295.15(28)	87.44(1)	26.44(30)	356.84(6)	77.23(30)	283.98(5)	298.29(22)	189.84(13)	382.90(5)	285.68(2)	366.36(6)	336.91(17)	272.78(13)	165.88(29)	302.86(26)	>600(0)	92.58(30)	221.82(1)	236.63(4)	242.23(15)	52.68(30)	240.08(15)	515.94(6)
	Best	47	159	797	47	162	788	50	179	51	172	926	50	200	978	49	168	893	52	196	53	225	1161	53	202
	Avg.	46.03	157.30	796.03	46.10	158.53	788.00	49.00	177.83	49.43	169.80	925.07	49.10	198.57	977.43	48.97	166.50	893.00	51.07	193.33	51.63	224.30	1161.00	51.53	199.10
	look4	46	157	797	46	160	788	50	178	50	172	926	50	199	978	49	166	894	51	195	53	225	1161	52	201
	7	0.999	0.95	0.90	0.999	0.95	0.90	0.999	0.95	0.999	0.95	0.90	0.999	0.95	0.90	0.999	0.95	0.90	0.999	0.95	0.999	0.95	0.90	0.999	0.95
	Instance	frb50-23-1			frb50-23-5			frb53-24-4		frb53-24-2			frb53-24-1			frb53-24-5			frb56-25-4		frb56-25-1			frb56-25-5	

TABLE 7. First experiment: 30 runs of each instance and algorithm limited to 600s of running time each.

B.Q. PINTO ET AL.

					BRKGA-IG*			BR	tKGA-LSQClique	
Instance	7	look4	Avg.	Best	Time (#100k4)	Avg. gen.	Avg.	Best	Time (#100k4)	Avg. gen.
frb56-25-2	0.999	52	51.43	53	280.08(12)	74.17	51.03	52	152.36(4)	42.75
	0.95	213	210.63	212	0.00(0)	0.00	213.80	217	198.41(29)	12.00
frb59-26-4	0.999	55	54.33	55	228.03(11)	52.45	53.73	55	322.46(4)	78.50
	0.95	230	228.27	232	340.83(6)	13.50	230.70	233	203.91(30)	10.87
frb59-26-1	0.999	55	54.17	56	346.35(7)	81.00	53.57	55	242.13(2)	55.50
	0.95	240	239.03	241	318.16(10)	11.30	240.90	244	156.25(30)	6.83
frb59-26-2	0.999	54	53.83	54	199.91(25)	46.44	53.60	54	201.10(18)	49.06
	0.95	235	234.57	237	351.97(16)	13.25	235.80	238	102.22(30)	4.37
frb59-26-5	0.999	54	53.80	55	198.84(22)	46.32	53.50	54	282.46(15)	68.40
	0.95	221	220.83	223	297.89(21)	13.91	221.90	224	89.80(30)	3.73

ach.	
e e	
tim	
ine	D
nn	
f ru	
)s c	1
60(,))
to	
ited	
lim	
un	
orit]	
alec	þ
pu	
ce a	
tan	
ins	
ach	
ofe	
ns e	
n (
3(
nent	
erin	
exp	4
rst	
L.	i I
२० १२)
ABL)	
Ē	

					BRKGA-IG*			BRJ	KGA-LSQClique	
stance	7	look4+	Avg.	Best	Time (#100k4)	Avg. gen.	Avg.	Best	Time (#100k4)	Avg. gen.
.9-999	0.999	74	74.07	75	535.71(30)	54.13	73.90	75	599.70(26)	71.65
	0.95	276	272.30	275	0.00(0)	0.00	276.63	279	434.55(30)	13.47
9	0.999	54	53.50	55	845.64(18)	36.78	53.33	55	662.55(16)	29.94
0.5	0.999	17	16.93	17	464.20(28)	19.00	16.97	18	355.98(28)	14.79
	0.90	27	26.83	27	761.12(25)	6.76	27.07	28	116.51(30)	1.20
	0.80	48	43.60	47	>600(0)	Ι	48.10	49	449.38(29)	5.35
0-40	0.999	86	85.90	87	779.23(23)	13.04	85.03	86	774.81(8)	16.00
	0.95	1835	1834.97	1835	885.90(28)	5.29	1835.57	1837	206.73(30)	1.13
	0.90	4	4.00	4	1.69(30)	6.30	4.00	4	1.78(30)	2.73
	0.60	2	7.00	2	3.40(30)	3.17	7.00	2	2.15(30)	1.10
	0.90	23	23.00	23	3.99(30)	8.17	23.00	23	1.94(30)	1.00
	0.70	30	30.00	30	36.86(30)	3.53	30.00	30	6.97(30)	1.13
	0.60	36	35.53	36	607.54(16)	32.00	36.00	36	420.33(30)	28.70
	0.50	44	43.80	44	486.85(24)	17.79	44.00	44	18.50(30)	1.00
0291_sey-	0.60	13	12.97	13	360.60(29)	31.52	13.00	13	9.49(30)	1.07
Liiasa	0.50	16	16.00	16	84.82(30)	5.27	16.00	16	13.56(30)	1.00
giantcompo	0.80	48	48.00	48	24.78(30)	2.40	48.00	48	9.09(30)	1.03
	0.60	60	60.00	60	50.92(30)	2.23	60.00	60	21.59(30)	1.07
	0.50	73	73.00	73	173.76(30)	4.33	73.00	73	26.58(30)	1.00

TABLE 9. First experiment: 30 runs of each instance and algorithm limited to 600s of running time each.

i.



FIGURE 3. Comparison between algorithms BRKGA-IG^{*} and BRKGA-LSQClique: fraction of the number of instances on which each algorithm outperforms the other for each value of γ . Considering all 150 instances and values of γ , BRKGA-LSQClique outperformed the original BRKGA-IG^{*} in 63.33% of the cases in terms of the time-to-target solution value.

Each heuristic was run 200 times for each value of the threshold γ . Next, the empirical probability distributions of the time taken by each heuristic to find the target solution value are plotted. To plot the empirical distribution for each heuristic, we followed the methodology proposed by Aiex *et al.* [2,3]. We associate a probability $p_i = (i - \frac{1}{2})/200$ with the *i*-th smallest running time t_i and plot the points (t_i, p_i) , for $i = 1, \ldots, 200$. The more to the left is a plot, the better is the algorithm corresponding to it.

Figures 4 and 5 illustrate the time-to-target plots for instance brock400_2 with $\gamma = 0.999$, 0.90, and 0.80 and for instance PGPgiantcompo with $\gamma = 0.90$, 0.80, and 0.50, respectively. We recall from Table 3 that BRKGA-IG^{*} obtained better results for brock400_2 with $\gamma = 0.90$. The plots in these figures show that BRKGA-IG^{*} performed better for $\gamma = 0.999$, with BRKGA-LSQClique becoming progressively better as γ decreases.

8. Concluding remarks

In this work, we showed that the exact enumeration algorithm QClique proposed by Ribeiro and Riveaux [25] can be hybridized with the biased random-key genetic algorithm BRKGA-IG^{*} developed by Pinto *et al.* [23] as a local search strategy to improve the quality of the solutions created by the decoder. The new decoder using an exact local search strategy gives rise to a matheuristic for solving the maximum cardinality quasi-clique problem based on a biased random-key genetic algorithm.

Computational experiments were performed on a set of benchmark instances, considering different thresholds.

The numerical results showed that although BRKGA-IG^{*} performed better for $\gamma = 0.999$, algorithm BRKGA-LSQClique becomes consistently better for smaller values of γ and considerably outperformed BRKGA-IG^{*} in terms of the time-to-target solution value in 63.33% of the 150 combinations of instances and values of γ .



FIGURE 4. Time-to-target plots for instance brock400_2. (a) $\gamma = 0.999$. (b) $\gamma = 0.90$. (c) $\gamma = 0.80$.



FIGURE 5. Time-to-target plots for instance PGP giantcompo. (a) $\gamma=0.90.$ (b) $\gamma=0.80.$ (c) $\gamma=0.50.$

S762

B.Q. PINTO ET AL.

Acknowledgements. The authors are thankful to three anonymous referees for their very constructive remarks that substantially contributed to improve the quality and the readability of this article. Work of Celso C. Ribeiro was partially supported by CNPq research grants 303958/2015-4 and 425778/2016-9 and by FAPERJ research grant E-26/202.854/2017. Work of José Angel Riveaux was supported by a CNPq scholarship. Work of Isabel Rosseti was partially supported by CNPq research grant 310624/2018-5. This work was also partially sponsored by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), under Finance Code 001.

References

- J. Abello, M. Resende and S. Sudarsky, Massive quasi-clique detection, edited by J. Abello and J. Vitter. In: Proceedings of the 5th Latin American Symposium on the Theory of Informatics. Springer-Verlag Berlin Heidelberg (2002) 598–612.
- R. Aiex, M. Resende and C.C. Ribeiro, Probability distribution of solution time in GRASP: an experimental investigation. J. Heuristics 8 (2002) 343–373.
- [3] R. Aiex, M. Resende and C.C. Ribeiro, TTTPLOTS: a Perl program to create time-to-target plots. Optim. Lett. 1 (2007) 355–366.
- [4] J.C. Bean, Genetic algorithms and random keys for sequencing and optimization. ORSA J. Comput. 2 (1994) 154-160.
- [5] BHOSLIB, Benchmarks with hidden optimum solutions for graph problems. http://networkrepository.com/ (2004). Online reference, last visited on June 7, 2018.
- [6] J.S. Brandão, T.F. Noronha, M.G.C. Resende and C.C. Ribeiro, A biased random-key genetic algorithm for single-round divisible load scheduling. Int. Trans. Oper. Res. 22 (2015) 823–839.
- [7] J.S. Brandão, T.F. Noronha, M.G.C. Resende and C.C. Ribeiro, A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems. Int. Trans. Oper. Res. 27 (2017) 1061–1077.
- [8] M. Brunato, H. Hoos and R. Battiti, On effectively finding maximal quasi-cliques in graphs, Learning and Intelligent Optimization edited by V. Maniezzo, R. Battiti and J.-P. Watson. In: Vol. 5313 of *Lecture Notes in Computer Science*. Springer, Berlin (2008) 41–55.
- [9] T.A. Davis and Y. Hu, The University of Florida sparse matrix collection. ACM Trans. Math. Softw. 38 (2011) 1–25.
- [10] FICO, FICO Xpress Optimization Suite 7.6. http://www.fico.com/en/products/fico-xpress-optimization-suite (2017).
- [11] J.F. Gonçalves and M.G.C. Resende, Biased random-key genetic algorithms for combinatorial optimization. J. Heuristics 17 (2011) 487–525.
- [12] J.F. Gonçalves, M.G.C. Resende and R.F. Toso, Biased and unbiased random key genetic algorithms: an experimental analysis. In: Abstracts of the 10th Metaheuristics International Conference. Singapore (2013).
- [13] H. Hoos and T. Stützle, Evaluation of Las Vegas algorithms Pitfalls and remedies, edited by G. Cooper and S. Moral. In: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence. Madison (1998) 238–245.
- [14] D.S. Johnson, Cliques, coloring, and satisfiability: second DIMACS implementation challenge. In: Vol. 26 of Dimacs Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, Providence (1996).
- [15] R.M. Karp, Reducibility among combinatorial problems. In: Complexity of Computer Computations, edited by R.E. Miller and J.W. Thatcher. Plenum, New York (1972) 85–103.
- [16] M. López-Ibánez, J. Dubois-Lacoste, T. Stützle and M. Birattari, The IRACE package: Iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011).
- [17] T.F. Noronha, M.G.C. Resende and C.C. Ribeiro, A biased random-key genetic algorithm for routing and wavelength assignment. J. Global Optim. 50 (2011) 503–518.
- [18] A.B. Oliveira, A. Plastino and C.C. Ribeiro, Construction heuristics for the maximum cardinality quasi-clique problem. In: Abstracts of the 10th Metaheuristics International Conference. Singapore (2013) 84.
- [19] F.M. Pajouh, Z. Miao and B. Balasundaram, A branch-and-bound approach for maximum quasi-cliques. Ann. Oper. Res. 216 (2014) 145–161.
- [20] G. Pastukhov, A. Veremyev, V. Boginski and O.A. Prokopyev, On maximum degree-based γ -quasi-clique problem: complexity and exact approaches. *Networks* **71** (2018) 136–152.
- [21] J. Pattillo, A. Veremyev, S. Butenko and V. Boginski, On the maximum quasi-clique problem. Discrete Appl. Math. 161 (2013) 244–257.
- [22] L. Pérez Cáceres, M. López-Ibáñez and T. Stützle, An analysis of parameters of IRACE. In: Proceedings of the 14th European Conference on Evolutionary Computation in Combinatorial Optimization. Vol. 8600 of Lecture Notes in Computer Science. Springer, Berlin (2014) 37–48.
- [23] B.Q. Pinto, C.C. Ribeiro, I. Rosseti and A. Plastino, A biased random-key genetic algorithm for solving the maximum quasiclique problem. Eur. J. Oper. Res. 271 (2018) 849–865.
- [24] M.G.C. Resende and C.C. Ribeiro, Biased-random key genetic algorithms: an advanced tutorial. In: Proceedings of the 2016 Genetic and Evolutionary Computation Conference – GECCO'16 Companion Volume. Association for Computing Machinery, Denver (2016) 483–514.
- [25] C.C. Ribeiro and J.A. Riveaux, An exact algorithm for the maximum quasi-clique problem. Int. Trans. Oper. Res. 26 (2019) 2199–2229.
- [26] R.A. Rossi and N.K. Ahmed, Coloring large complex networks. Soc. Network Anal. Min. 4 (2014) 1–37.

- [27] R.A. Rossi and N.K. Ahmed, The network data repository with interactive graph analytics and visualization. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (2015) 4292–4293.
- [28] E.C. Sewell, An improved algorithm for exact graph coloring, Cliques, Coloring, and Satisfiability, edited by D.S. Johnson and M.A. Trick. In: Vol. 26 of 2nd DIMACS Implementation Challenge. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society (1996) 359–373.
- [29] W. Spears and K. de Jong, On the virtues of parameterized uniform crossover, edited by R. Belew and L. Booker. Proceedings of the Fourth International Conference on Genetic Algorithms. Morgan Kaufman, San Mateo (1991) 230–236.
- [30] R.F. Toso and M.G.C. Resende, A C++ application programming interface for biased random-key genetic algorithms. Optim. Methods Softw. 30 (2015) 81–93.
- [31] A. Veremyev, O.A. Prokopyev, S. Butenko, and E.L. Pasiliao, Exact MIP-based approaches for finding maximum quasi-clique and dense subgraph. Comput. Optim. App. 64 (2016) 177–214.

APÊNDICE C – The minimum quasi-clique partitioning problem: Complexity, formulations, and a greedy randomized heuristic

The minimum quasi-clique partitioning problem: Complexity, formulations, and a greedy randomized heuristic

Rafael A. Melo^{*} Celso C. Ribeiro[§] Jose A. Riveaux[¶]

December 17, 2021

Abstract

Given a simple graph G = (V, E) and a real constant $\gamma \in (0, 1]$, a γ -clique (or γ -quasi-clique) is a subset $V' \subseteq V$ inducing a subgraph with edge density at least γ . The minimum quasi-clique (or γ -clique) partitioning problem (MQCPP) consists in partitioning the vertices of the graph in γ -cliques so as to minimize the number of elements in the partition. In this paper, we formally introduce the minimum quasi-clique partitioning problem. We show by using a reduction from the unweighted maximum cut problem that even deciding whether a graph can be partitioned into two γ -cliques is NP-complete. This result contrasts with those for the well-known clique partitioning problem, whose decision version is polynomially solvable for bipartition. Additionally, we propose four integer programming formulations and a multi-start greedy randomized heuristic to provide initial feasible solutions for MQCPP. Computational experiments show that two formulations that employ the principles of representatives outperform the others when it comes to the best obtained solutions and the number of instances solved to optimality within the imposed time limit. Furthermore, the results also demonstrate that the instances with medium values of γ are more challenging for the proposed formulations than those with larger or lower values.

Keywords: quasi-clique partitioning; maximum quasi-clique; computational complexity; integer programming; combinatorial optimization; network clustering.

1 Introduction

1.1 Preliminaries

Let G = (V, E) be a simple undirected graph with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E \subset V \times V$. *G* is a complete graph if there is an edge in *E* connecting every two different vertices in *V*. A clique is a complete subgraph of a graph. The complement of *G* is the graph $\overline{G} = (V, \overline{E})$ with $\overline{E} = \{uv \mid uv \notin E\}$. Denote by anti-edge (or non-edge) of *G* an edge in \overline{E} . Besides, given a subset $S \subseteq V$, define $\overline{S} = V \setminus S$.

The graph G[S] induced in G by $S \subseteq V$ is that with vertex set S and edge set formed by all edges of E with both endpoints in S. For any $S \subseteq V$, the subset $E(S) \subseteq E$ is formed by all edges of G with both endpoints in S. Also, consider $W \subseteq V$ such that $S \cap W = \emptyset$, and denote by E(S, W) the set of all edges in G with one endpoint in S and another in W. A partitioning of a graph (or graph partition) is a partition of its vertex set.

The density of a graph G is defined as $d(G) = \frac{|E|}{|V| \cdot (|V|-1)/2} = \frac{2|E|}{|V| \cdot (|V|-1)}$. The degree of a vertex $v \in V$, $deg_G(v)$, is given by the number of vertices adjacent to v in G. Given a graph G = (V, E) and a threshold $\gamma \in (0, 1]$, a γ -clique (or γ -quasi-clique) is any subset $C \subseteq V$ such that the density of G[C] is greater than or equal to γ . A γ -clique C is maximal if there is no other γ -clique C' such that $C \subset C'$. The minimum

^{*}Institute of Computing, Universidade Federal da Bahia, Salvador, BA 40170-115, Brazil (rafael.melo@ufba.br).

 $^{^{\$}}$ Institute of Computing, Universidade Federal Fluminense, Niterói, RJ 24210-346, Brazil (celso@ic.uff.br). Corresponding author.

[¶]Institute of Computing, Universidade Federal Fluminense, Niterói, RJ 24210-346, Brazil (jangel.riveaux@gmail.com).

quasi-clique (or γ -clique) partitioning problem (MQCPP) consists in finding the minimum number \mathcal{K} of quasi-cliques in which G can be partitioned.

1.2 Literature review

Problems related to encountering dense subgraphs in networks have appeared in several applications, including telecommunications (Abello et al., 2002), biology (Spirin and Mirny, 2003), and as approaches for solving other challenging combinatorial optimization problems (Blum et al., 2021). Among the important problems of encountering large dense subgraphs, we can highlight the maximum clique problem and the maximum quasi-clique problem. Besides, partitioning graphs into dense subgraphs finds applications in several areas, such as bioinformatics (Hu et al., 2005), quantum computing (Verteletskyi et al., 2020), and data mining (Glaria et al., 2021). The minimum quasi-clique partitioning problem lies in this family of problems.

The maximum clique problem is a classical NP-hard combinatorial optimization problem that consists in obtaining a maximum cardinality clique in a graph. Theoretical and computational aspects of the problem and its variants have been extensively studied throughout the last decades (Pardalos and Xue, 1994; Bomze et al., 1999; Östergård, 2002; Wu and Hao, 2015; Li et al., 2017). The clique partitioning problem and its derivations also received a lot of attention in the literature (Grötschel and Wakabayashi, 1990; De Amorim et al., 1992; Oosten et al., 2001; Gramm et al., 2009; Brimberg et al., 2017). Observe that the clique partitioning problem is strongly related to graph coloring (Jensen and Toft, 2011), as a clique partition in a graph determines a vertex coloring (Malaguti and Toth, 2010) in its complement. However, the concept of clique may be considered too restrictive for applications such as network analysis (Pattillo et al., 2013) and, thus, different clique relaxations have been studied, such as quasi-cliques.

Problems consisting of obtaining maximum quasi-cliques, or at least maximal ones, attracted the attention of several researchers during the last decades. Abello et al. (2002) proposed an approach that employs preprocessing and a greedy randomized adaptive search procedure (GRASP) metaheuristic for obtaining quasi-cliques in massive sparse graphs. Tsourakakis et al. (2013) defined the problem of obtaining what they call optimal quasi-cliques. They proposed a greedy approximation algorithm and a local search heuristic for obtaining such structures. Pattillo et al. (2013) studied the maximum quasi-clique problem (MQCP). The authors showed that the problem is NP-hard for any $0 < \gamma < 1$, proposed analytical upper bounds, and presented integer programming formulations. Veremyev et al. (2016) proposed new integer programming formulations for MQCP and compared them with those presented in Pattillo et al. (2013), both theoretically and computationally. Pinto et al. (2018) elaborated a biased random-key genetic algorithm (BRKGA) for MQCP. Ribeiro and Riveaux (2019) provided an exact backtracking-based algorithm for MQCP. Zhou et al. (2020) described a memetic algorithm for MQCP that employs tabu search for local improvement and opposition-based learning (OBL) to attempt enhancing its search mechanism. Pinto et al. (2021) proposed a BRKGA metaheuristic with an exact local search strategy that combines the methods presented in Pinto et al. (2018) and Ribeiro and Riveaux (2019). Marinelli et al. (2021) proposed a new formulation for MQCP with an exponential number of variables to provide strong linear relaxation bounds using column generation. The problem of obtaining maximal degree-based quasi-cliques, i.e., those in which each vertex has degree at least $\gamma(|V|-1)$, was addressed in Sanei-Mehri et al. (2021). The authors proved the NP-completeness of determining if a given degree-based quasi-clique is maximal and proposed a heuristic to the problem of obtaining the k largest degree-based quasi-cliques in a graph.

Analogously to the clique partitioning problem, which involves partitioning the graph into cliques, the quasi-clique partitioning problem consists of partitioning the graph into quasi-cliques. As it was observed in the literature, the problem of clustering networks can be modeled as minimum clique partition problems, since all elements in the cluster are tightly related. However, due to the strong restrictiveness of cliques, it may be worthy modeling clusters by k-cores (maximal subgraphs with degree at least k) or clique relaxations such as s-plexes (subgraphs in which every vertex is nonadjacent to at most s-1 other vertices), s-clubs (sets of vertices inducing subgraphs of diameter at most s), or quasi-cliques (Lee et al., 2010; Verma and Butenko, 2013). To the best of our knowledge, differently from the minimum clique partitioning, the minimum quasi-clique partitioning problem was not yet formally addressed in the literature despite its potentially large applicability. As far as we know, the only work considering the quasi-clique partitioning of a graph is

Basu et al. (2014). The authors proposed a game theoretical approach for partitioning a graph into (λ, γ) cliques (which are quasi-cliques characterized not only by their densities, but also by the degrees of their vertices) associated with communities. However, the problem is not analyzed from an optimization point of view. Instead, the partitions are evaluated according to network quality measures regarding the encountered communities.

1.3 Contributions and organization

The principal contributions of our paper can be summarized as follows. Firstly, we formally introduce the minimum quasi-clique partitioning problem (MQCPP) and show that it is NP-hard even when the optimal solution is equal to two, i.e., when the instance admits a partition into two quasi-cliques. Secondly, we present and compare computationally four compact integer programming formulations (with polynomial numbers of variables and constraints) for the problem. Thirdly, we propose a multi-start greedy randomized heuristic to provide initial feasible solutions for the formulations.

The remainder of this paper is organized as follows. The computational complexity of MQCPP is addressed in Section 2. Integer programming formulations are presented in Section 3. Section 4 shows the proposed multi-start greedy randomized heuristic. The computational experiments are summarized in Section 5. Concluding remarks are discussed in the last section.

2 Computational complexity

In this section, we show that the minimum quasi-clique partitioning problem (MQCPP) is NP-hard even for a very restrictive case, namely, when the number of quasi-cliques in the optimal solution is equal to two. To do so, consider the decision version of MQCPP, which can be defined as follows:

Quasi-clique partitioning problem (QCPP) Instance: Graph G = (V, E), threshold value $\gamma \in (0, 1]$, and an integer $\mathcal{K} \geq 2$. Question: Is there a partition of G into $\mathcal{K} \gamma$ -cliques?

Observe that, when $\gamma = 1$, QCPP corresponds to the well-known clique partitioning problem (CPP), which was shown to be NP-complete for $\mathcal{K} > 2$ (Karp, 1972). Thus, QCPP is also NP-complete for $\mathcal{K} > 2$. It should be noticed, however, that CPP can be solved in polynomial time (and thus is in P) for $\mathcal{K} = 2$, since it amounts to check whether the complement \overline{G} of G is bipartite or not. In the following, we show that QCPP is NP-complete when $\gamma \in (0, 1)$ even for $\mathcal{K} = 2$. In this direction, consider the following special case of QCPP:

Quasi-clique bipartitioning problem (QCBP) Instance: Graph G = (V, E) and threshold value $\gamma \in (0, 1)$. Question: Is there a partition of G into two γ -cliques?

In what follows, Theorem 1 shows that QCBP is NP-complete. The proof uses a reduction from the NP-complete decision version of the unweighted maximum cut problem (Garey et al., 1974), also known as simple maximum cut problem, which can be stated as:

Unweighted maximum cut problem (MCP) Instance: Graph G = (V, E) and integer L. Question: Is there $P \subset V$ such that $|E(P, \overline{P})| \geq L$?

Theorem 1. QCBP is NP-complete.

Proof. We first show that QCBP belongs to NP. Given any solution for QCBP, represented by the partition (S, \overline{S}) with $S \subset V$ and $S \neq \emptyset$, the densities of the subgraphs G[S] and $G[\overline{S}]$ induced in G by S and \overline{S} ,

respectively, can be computed in linear-time O(|V| + |E|). Therefore, QCBP is in NP.

a

We now present a polynomial transformation from the unweighted maximum cut problem to the quasiclique bipartitioning problem, i.e., MCP \propto QCBP. Therefore, as the former is known to be NP-complete, this implies that the latter is also NP-complete, as we already showed that it belongs to NP.

The polynomial transformation works as follows. Given an input graph G = (V, E) for MCP, we generate for QCBP an input graph G' = (V', E') with $2n(n^2 + 1) = 2n^3 + 2n$ vertices such that:

$$V' = \{w_1^0, \dots, w_1^{n^2}\} \cup \dots \cup \{w_n^0, \dots, w_n^{n^2}\} \cup \{\breve{w}_1^0, \dots, \breve{w}_1^{n^2}\} \cup \dots \cup \{\breve{w}_n^0, \dots, \breve{w}_n^{n^2}\}$$

and
$$E' = \{w_h^i w_k^j, \breve{w}_h^i \breve{w}_k^j: h, k = 1, \dots, n; \ j = 0, \dots, n^2; \ i = 1, \dots, n^2; \ \text{with} \ i \neq j \text{ or } h \neq k\}$$
$$\cup \{w_h^i \breve{w}_k^j, \breve{w}_h^i w_k^j: h, k = 1, \dots, n: k \neq h; \ j = 0, \dots, n^2; \ i = 1, \dots, n^2\}$$
$$\cup \{w_h^0 \breve{w}_k^0, \breve{w}_h^0 w_k^0: h, k = 1, \dots, n: v_h v_k \in E\}.$$

We observe that, for every h = 1, ..., n, each vertex $w_h^i \in V'$, with $i = 1, ..., n^2$, is adjacent to all other vertices in V', with the exception of vertices $\breve{w}_h^j \in V'$, with $j = 0, ..., n^2$. Additionally, notice that $\{w_h^0 : h = 1, ..., n\}$ and $\{\breve{w}_h^0 : h = 1, ..., n\}$ are independent sets. Observe that G' can be obtained from G in polynomial time as it has $O(n^3)$ vertices and $O(n^6)$ edges. This construction is illustrated in Example 1.

To finalize the polynomial transformation, given the integer L for MCP, define γ for QCBP as

$$\gamma = \frac{2\left[\binom{n^3}{2} + n^4 + L\right]}{n(n^2 + 1)[n(n^2 + 1) - 1]}.$$
(1)

Propositions 1 and 2, which will be detailed in the following, show that there is a cut of size greater than or equal to L in G = (V, E) if and only if there is a γ -clique bipartition in G' = (V', E'). Thus, implying that QCBP is NP-complete.

Example 1. Figure 1(a) exemplifies an input graph for MCP, while Figure 1(b) shows the corresponding graph G' constructed from the one in Figure 1(a). To simplify the illustration, the clique formed by the vertices $\{w_h^i : i = 1, ..., n^2\}$ is represented by a single super-vertex W_h in Figure 1(b), for every h = 1, ..., n. Analogously, the clique formed by the vertices $\{\check{w}_h^i : i = 1, ..., n^2\}$, for every h = 1, ..., n, is represented by a unique super-vertex \check{W}_h .

$$\triangle$$

Proposition 1. Given G = (V, E) and G' = (V', E') as defined in Theorem 1, if there exists a cut of G = (V, E) with size greater than or equal to L, then there exists a γ -clique bipartitioning of G' = (V', E').

Proof. Assume there is a subset $P \subset V$ defining a cut (P, \overline{P}) for which $|E(P, \overline{P})| \geq L$. Define the bipartition (S, \overline{S}) , with

$$S = \{w_h^0, \dots, w_h^{n^2} : h = 1, \dots, n; v_h \in P\} \cup \{\breve{w}_h^0, \dots, \breve{w}_h^{n^2} : h = 1, \dots, n; v_h \in \bar{P}\}.$$

Notice that $|S| = n(n^2 + 1)$. Therefore, given the construction of G', $S \setminus \{w_h^0, \check{w}_h^0 : h = 1, ..., n\}$ induces a clique K_{n^3} with n^3 vertices in G' as the anti-edge associated with the pair w_h^i and \check{w}_h^i is not present for any $i \in \{1, ..., n^2\}$ and $h \in \{1, ..., n\}$. Thus, the vertices of such set are all pairwise connected in G', guaranteeing the existence of $\binom{n^3}{2}$ edges in E'(S). Also due to the construction of G', all the vertices in the subsets $W^0 = \{w_h^0 : h = 1, ..., n; v_h \in P\}$ and $\check{W}^0 = \{\check{w}_h^0 : h = 1, ..., n; v_h \in \bar{P}\}$ are adjacent to all the vertices in K_{n^3} , ensuring the existence of $n \cdot n^3 = n^4$ edges in E'(S). Moreover, if there are at least Ledges between P and \bar{P} in G, then there must be L or more edges in $E'(W^0 \cap S, \check{W}^0 \cap S)$, corresponding to the edges $w_h^0 \check{w}_{h'}^0$ such that $v_h v_{h'} \in E$. Consequently, as these sets of guaranteed edges are disjoint,



Figure 1: Example of the two graphs involved in the polynomial transformation MCP \propto QCBP.

 $|E'(S)| \ge {\binom{n^3}{2}} + n^4 + L$, implying that

$$d(G'[S]) = \frac{2|E'(S)|}{|S| \cdot (|S| - 1)} \ge \frac{2\left\lfloor \binom{n^3}{2} + n^4 + L \right\rfloor}{n(n^2 + 1)[n(n^2 + 1) - 1]}$$

Notice that using the same arguments in a symmetric way, it follows that $|\bar{S}| = |S|$ and $|E'(\bar{S})| = |E'(S)|$. Hence,

$$d(G'[\bar{S}]) = \frac{2|E'(\bar{S})|}{|\bar{S}| \cdot (|\bar{S}| - 1)} = \frac{2|E'(S)|}{|S| \cdot (|S| - 1)} \ge \frac{2[\binom{n^3}{2} + n^4 + L]}{n(n^2 + 1)[n(n^2 + 1) - 1]}.$$

Therefore, G'[S] and $G'[\overline{S}]$ are γ -cliques. This is illustrated in Example 2.

~

Example 2. Consider again the graph illustrated in Figure 1(a), for which n = 3, and assume L = 2. We show that if there exists a cut of G with size two, then equation (1) implies that there will be a γ -clique bipartitioning in G' for

$$\gamma = \frac{2 \cdot \left[\binom{3^3}{2} + 3^4 + 2\right]}{3 \cdot (3^2 + 1) \cdot [3 \cdot (3^2 + 1) - 1]} = \frac{868}{870}$$

Notice that the set $P = \{v_1, v_3\}$ defining the cut $(\{v_1, v_3\}, \{v_2\})$, illustrated in Figure 2, is a solution for MCP. Given the construction of G', taking the partition $(\{v_1, v_3\}, \{v_2\})$, we obtain $S = \{w_1^0, w_3^0, \breve{w}_2^0\} \cup W_1 \cup W_3 \cup \breve{W}_2$ and $\bar{S} = \{\breve{w}_1^0, \breve{w}_3^0, w_2^0\} \cup \breve{W}_1 \cup \breve{W}_3 \cup W_2$, with their corresponding edges illustrated in the continuous lines of Figures 3(a) and 3(b), respectively. Both the induced subgraphs G'[S] and $G'[\bar{S}]$ have 30 vertices and $\binom{3^3}{2} + 3^4 + 2 = 434$ edges. Thus, their densities are equal to $\frac{2\cdot434}{30\cdot29} = \frac{868}{870} = \gamma$.

	Λ
/	`\
-	_



Figure 2: Partition $(\{v_1, v_3\}, \{v_2\})$ defining a cut for G.



Figure 3: Bipartition of graph G'.

Proposition 2. Given G = (V, E) and G' = (V', E') as defined in Theorem 1, if there exists a bipartition of G' in γ -cliques S and \overline{S} , then there is a cut of G = (V, E) with size greater than or equal to L.

Proof. Let $S \subset V'$ and denote by (S, \overline{S}) a bipartition of V' such that G'[S] and $G'[\overline{S}]$ are γ -cliques. Without loss of generality, suppose that $|S| \ge |V'|/2$. Since G' has $2n(n^2 + 1)$ vertices and because of the quasi-hereditary property of γ -cliques (Pattillo et al., 2013), S must contain a γ -clique S' of G' with size $|V'|/2 = n(n^2 + 1) = n^3 + n$. Consequently, equation (1) implies that the number of edges in G'[S'] is greater than or equal to $\binom{n^3}{2} + n^4 + L$.

Notice that, since $|S'| = n^3 + n$ and the number of edges in G'[S'] is greater than or equal to $\binom{n^3}{2} + n^4 + L$, the maximum number of anti-edges in G'[S'] is

$$\binom{n^3+n}{2} - \left[\binom{n^3}{2} + n^4 + L\right] = \binom{n}{2} - L.$$

Lemmas 1, 2, 3, and 4, which will be detailed next, show, respectively, that:

- (a) S' does not simultaneously contain one vertex from W_h and another from \check{W}_h for any $h = 1, \ldots, n$;
- (b) for every h = 1, ..., n, $W_h \cap S' \neq \emptyset$ and $|W_h \cap S'| \ge n^2 n$ if and only if $\breve{W}_h \cap S' = \emptyset$;
- (c) $|(W_1 \cup \ldots \cup W_n \cup \breve{W}_1 \cup \ldots \cup \breve{W}_n) \cap S'| \le n^3$; and
- (d) for every h = 1, ..., n, either $W_h \cup \{w_h^0\} \subset S'$ or $\breve{W}_h \cup \{\breve{w}_h^0\} \subset S'$.

Notice that the construction of G' together with point (d) (given by Lemma 4) imply that, for every $h = 1, \ldots, n$, exactly one of the sets $W_h \cup \{w_h^0\}$ or $\check{W}_h \cup \{\check{w}_h^0\}$ is in S'. Thus, every one of the n^3 vertices in $(W_1 \cup \ldots \cup W_n \cup \check{W}_1 \cup \ldots \cup \check{W}_n) \cap S'$ is adjacent to every other vertex in S' (we recall that there are $n^3 + n$ of them), ensuring $\binom{n^3}{2} + n^4$ of the edges in G'[S']. Because G'[S'] has at least $\binom{n^3}{2} + n^4 + L$ edges, L or more edges must exist between the n vertices of $(w_1^0, \ldots, w_n^0, \check{w}_1^0, \ldots, \check{w}_n^0) \cap S$. Since both sets $\{w_h^0 : h = 1, \ldots, n\}$ and $\{\check{w}_h^0 : h = 1, \ldots, n\}$ are independent, these edges can only exist between the vertices of $\{w_h^0 : h = 1, \ldots, n\} \cap S'$ and $\{\check{w}_h^0 : h = 1, \ldots, n\} \cap S'$, and correspond to those built from the edges in the input graph G of MCP. Consequently, considering $P = \{v_h \in V : w_h^0 \in S'\}$, it follows that the partition (P, \bar{P}) defines a cut with at least L edges in G for MCP.

Lemma 1. Let S' be defined as in Proposition 2. S' does not simultaneously contain one vertex from W_h and another from \check{W}_h , for any h = 1, ..., n.

Proof. Observe that the 2n vertices $w_1^0, \ldots, w_n^0, \check{w}_1^0, \ldots, \check{w}_n^0$ are the only vertices of V' that do not belong to $W_1 \cup \ldots \cup W_n \cup \check{W}_1 \cup \ldots \cup \check{W}_n$. This implies that $|(W_1 \cup \ldots \cup W_n \cup \check{W}_1 \cup \ldots \cup \check{W}_n) \cap S'| \ge n^3 - n$. Notice that the set $(W_1 \cup \ldots \cup W_n \cup \check{W}_1 \cup \ldots \cup \check{W}_n) \cap S'$ can be seen as the union of subsets $(W_i \cup \check{W}_i) \cap S'$ for $i = 1, \ldots, n$, as $W_1, \ldots, W_n, \check{W}_1, \ldots, \check{W}_n$ are disjoint sets. Let $\Gamma \subseteq \{1, \ldots, n\}$ contain every index ℓ such that S' contains vertices from both W_ℓ and \check{W}_ℓ . Besides, let $\bar{\Gamma} = \{1, \ldots, n\} \setminus \Gamma$ be the set containing every index ℓ such that S' contains vertices from W_ℓ if and only if S' does not contain vertices from \check{W}_ℓ .

Firstly, we show that if $\Gamma \neq \emptyset$, then there must be at least one $\ell' \in \Gamma$ for which $|(W_{\ell'} \cup \check{W}_{\ell'}) \cap S'| \ge n^2 - n$. Suppose by contradiction that $|(W_{\ell} \cup \check{W}_{\ell}) \cap S'| < n^2 - n$ for every $\ell \in \Gamma$. Let $p = |\bar{\Gamma}|$ and $q = |\Gamma|$, and notice that p+q = n. Thus, the number of vertices in $(W_1 \cup \ldots \cup W_n \cup \check{W}_1 \cup \ldots \cup \check{W}_n) \cap S'$ is at most $pn^2 + q(n^2 - n - 1)$. To see why, notice that there can be at most n^2 vertices for each $\ell \in \bar{\Gamma}$, which are the n^2 vertices of the corresponding set (either W_{ℓ} or \check{W}_{ℓ}), and at most $(n^2 - n - 1)$ vertices for each $\ell \in \Gamma$, given the supposition. However, as $q = |\Gamma| \ge 1$, it follows that $pn^2 + q(n^2 - n - 1) \le (n - 1)n^2 + (n^2 - n - 1) = n^3 - n - 1 < n^3 - n$. Therefore, we reach a contradiction as there must exist $\ell' \in \Gamma$ such that $|(W_{\ell'} \cup \check{W}_{\ell'}) \cap S'| \ge n^2 - n$.

We now show that if $\Gamma \neq \emptyset$, then the number of anti-edges in G'[S'] is strictly greater than $\binom{n}{2} - L$, implying that G'[S'] cannot be a γ -clique. Recall that, given the construction of G', there are no edges between the vertices of $W_{\ell'}$ and those of $\breve{W}_{\ell'}$, i.e., $E'(W_{\ell'}, \breve{W}_{\ell'}) = \emptyset$. All the anti-edges between the vertices of $W_{\ell'} \cap S'$ and $\breve{W}_{\ell'} \cap S'$ will be anti-edges in G'[S'], and their number can be computed as $|W_{\ell'} \cap S'| \cdot |\breve{W}_{\ell'} \cap S'|$. Since $|(W_{\ell'} \cup \breve{W}_{\ell'}) \cap S'| \ge n^2 - n$, the number of anti-edges in G'[S'] between the vertices of $W_{\ell'}$ and $\breve{W}_{\ell'}$ is at least $\{\min xy \mid x + y \ge n^2 - n, x \ge 1, y \ge 1\}$, whose solution is $n^2 - n - 1$ and is achieved when either x = 1 or y = 1 (this can be checked by inspection). This implies that there are at least $n^2 - n - 1 > \binom{n}{2} - L$ anti-edges in $G'[(W_h \cup \breve{W}_h) \cap S']$. Thus, Γ has to be empty, and the result follows.

Lemma 2. Let S' be defined as in Proposition 2. For every h = 1, ..., n, $W_h \cap S' \neq \emptyset$ and $|W_h \cap S'| \ge n^2 - n$ if and only if $\breve{W}_h \cap S' = \emptyset$.

Proof. If $W_h \cap S' \neq \emptyset$ then, by Lemma 1, no vertex of \check{W}_h belongs to S', i.e., $\check{W}_h \cap S' = \emptyset$. On the other hand, if $\check{W}_h \cap S' = \emptyset$ then, for every $\ell = 1, \ldots n$ with $\ell \neq h$, at most n^2 vertices of either W_ℓ or \check{W}_ℓ may belong to S', adding up to a total of at most $n^2(n-1) = n^3 - n^2$ vertices. Therefore, as $|S'| = n^3 + n$, at least $n^2 - n$ vertices of W_h must belong to S', i.e., $S' \cap W_h \neq \emptyset$.

Lemma 3. Let S' be defined as in Proposition 2. $|(W_1 \cup \ldots \cup W_n \cup \breve{W}_1 \cup \ldots \cup \breve{W}_n) \cap S'| \leq n^3$.

Proof. We know from Lemma 2 that, for each h = 1, ..., n, only vertices of either W_h or \check{W}_h may belong to S'. Since the sets W_h and \check{W}_h have exactly n^2 vertices each, the number of vertices in $|(W_1 \cup ... \cup W_n \cup \check{W}_1 \cup ... \cup \check{W}_n) \cap S'|$ is not greater than $n \cdot n^2 = n^3$.

Lemma 4. Let S' be defined as in Proposition 2. For every h = 1, ..., n, either $W_h \cup \{w_h^0\} \subset S'$ or $\check{W}_h \cup \{\check{w}_h^0\} \subset S'$.

Proof. Notice that Lemma 2 implies that either $|W_h \cap S'| \ge n^2 - n$ or $|\check{W}_h \cap S'| \ge n^2 - n$. Assume by contradiction that there is some $h \ge 1$ such that either $w_h^0 \in S'$ and $\check{W}_h \cap S' \ne \emptyset$ or $\check{w}_h^0 \in S'$ and $W_h \cap S' \ne \emptyset$. This, together with the fact that $E'(w_h^0, \check{W}_h) = E'(\check{w}_h^0, W_h) = \emptyset$, implies that the number of anti-edges in G[S'] is at least $n^2 - n > \binom{n}{2} - L$. Therefore, the number of edges in G'[S'] is strictly smaller than $\binom{n^3}{2} + n^4 + L$, which contradicts the fact that G'[S'] is a γ -clique. Moreover, given Lemma 3 and the fact that $|S'| = n(n^2 + 1) = n^3 + n$, it follows that for every $h = 1, \ldots, n$ either $W_h \cup \{w_h^0\} \subset S'$ (contributing with $n^2 + 1$ vertices each) or $\check{W}_h \cup \{\check{w}_h^0\} \subset S'$ (contributing with $n^2 + 1$ vertices each).

As QCBP is a special case of QCPP, Corollary 1 holds as a consequence of Theorem 1.

Corollary 1. QCPP is NP-complete.

Thus, Corollary 1 implies that the minimum quasi-clique partition problem (MQCPP) is NP-hard even when its optimal solution is equal to two.

3 Integer programming formulations

Integer programming approaches have been successfully applied recently to a variety of challenging graphrelated problems (Dell'Amico et al., 2022; Melo et al., 2021a; Marzo et al., 2022). In this section we present four compact integer programming formulations for the minimum quasi-clique partitioning problem (MQCPP). The first formulation, presented in Section 3.1, models quasi-cliques and graph partitions in natural ways, following ideas similar to those used in, respectively, Pattillo et al. (2013) and Méndez-Díaz and Zabala (2006). The second formulation, described in Section 3.2, relies on the idea of quasi-clique size decompositions (Veremyev et al., 2016). The third formulation, detailed in Section 3.3, uses the principles of formulations by representatives (Campêlo et al., 2004; Frota et al., 2010) to model partitions in an attempt to reduce symmetry. The fourth formulation, given in Section 3.4, combines both the ideas of quasi-clique size decompositions and formulations by representatives.

In the remainder of the section, denote the vertex set of G = (V, E) by $V = \{1, ..., n\}$ for the sake of notation simplification. Additionally, define UB to be any valid upper bound on the number of quasi-cliques in an optimal partition. Note that any feasible solution for the problem can be used to determine UB. Besides, define UB_k to be an upper bound on the size of any quasi-clique in the graph. The following bound can be used (Pattillo et al., 2013):

$$UB_k = \left\lfloor \frac{1}{2} + \frac{1}{2}\sqrt{1 + 8\frac{|E|}{\gamma}} \right\rfloor.$$
(2)

3.1 Standard formulation

A standard (or natural) integer programming formulation for MQCPP can be established using the decision variables described in the following. For every $i \in \{1, ..., UB\}$, define the binary variable

$$y_i = \begin{cases} 1, \text{ if a quasi-clique indexed by } i \text{ belongs to the partition,} \\ 0, \text{ otherwise.} \end{cases}$$

Besides, for every $i \in \{1, \ldots, UB\}$ and $v \in V$, consider the binary variable

$$x_{iv} = \begin{cases} 1, \text{ if vertex } v \text{ belongs to the quasi-clique } i, \\ 0, \text{ otherwise.} \end{cases}$$

Furthermore, for any $i \in \{1, \ldots, UB\}$ and $u, v \in V$, with u < v, define the binary variable

 $w_{iuv} = \begin{cases} 1, \text{ if vertices } u \text{ and } v \text{ are both in quasi-clique } i, \\ 0, \text{ otherwise.} \end{cases}$

Thus, MQCPP can be formulated as the following integer program:

$$(STD) \quad \min \sum_{i=1}^{UB} y_i \tag{3}$$
$$\sum_{i=1}^{UB} x_{iv} = 1, \qquad \forall v \in V, \qquad (4)$$

$$\sum_{i=1}^{N} x_{iv} = 1, \qquad \forall v \in V, \qquad (4)$$
$$x_{iv} \le y_i, \qquad \forall i = 1, \dots, UB, \quad v \in V, \qquad (5)$$

$$\begin{aligned} x_{iv} &\leq y_i, & \forall i = 1, \dots, UB, \ v \in V, \\ x_{iu} + x_{iv} &\leq w_{iuv} + 1, & \forall i = 1, \dots, UB, \ u, v \in V : u < v, \end{aligned}$$

$$w_{iuv} \le x_{iu}, \qquad \qquad \forall i = 1, \dots, UB, \ u, v \in V : u < v, \qquad (7)$$

$$w_{iuv} \le x_{iv}, \qquad \forall i = 1, \dots, UB, \ u, v \in V : u < v, \qquad (8)$$

$$\sum_{u \in V} \sum_{\substack{v \in V \\ u < v, uv \in E}} w_{iuv} \ge \gamma \cdot \sum_{u \in V} \sum_{\substack{v \in V \\ u < v}} w_{iuv}, \qquad \forall i = 1, \dots, UB, \qquad (9)$$

$$y_{i} \ge y_{i+1}, \qquad \forall i = 1, \dots, UB - 1, \qquad (10)$$

$$y_{i} \in \{0, 1\}, \qquad \forall i = 1, \dots, UB, \qquad (11)$$

$$x_{iv} \in \{0, 1\}, \qquad \forall i = 1, \dots, UB, \quad v \in V, \qquad (12)$$

$$\forall i = 1, \dots, UB, \quad v \in V,$$

$$\forall i = 1, \dots, UB, \quad v \in V : u < v.$$

$$\forall i = 1, \dots, UB, \quad u, v \in V : u < v.$$

$$(12)$$

The objective function (3) minimizes the number of quasi-cliques in the partition. Constraints (4) establish that each vertex belongs to exactly one quasi-clique. Constraints (5) guarantee that a vertex may belong to a quasi-clique only if this quasi-clique belongs to the partition. Constraints (6)-(8) enforce that if both vertices u and v are part of the quasi-clique indexed by i then the existence or not of the edge between them will affect the density of this quasi-clique. Next, constraints (9) enforce that the density of every quasi-clique should be greater than or equal to γ . Constraints (10) are symmetry breaking constraints. Finally, constraints (11)-(13) ensure the integrality requirements of the variables.

We remark that whenever one wants to avoid equality constraints in the formulation, constraints (4) can be replaced by

$$\sum_{i=1}^{UB} x_{iv} \ge 1, \qquad \forall v \in V, \tag{14}$$

$$\forall v \in V, \quad i, i' = 1, \dots, UB : i < i'.$$
 (15)

3.2 Formulation using quasi-clique size decompositions

u

110

 x_i

In this section, we present a formulation using the idea of Veremyev et al. (2016) to model the quasi-cliques in each element of the partition. The authors showed through computational experiments that such an approach was effective for finding maximum quasi-cliques in sparse graphs. Besides the y and x variables already defined in Section 3.1, the formulation uses the following decision variables. Consider, for every $i \in \{1, \ldots, UB\}$ and $k \in \{1, \ldots, UB_k\}$, the binary variable

$$z_{ik} = \begin{cases} 1, \text{ if a quasi-clique indexed by } i \text{ has size } k, \\ 0, \text{ otherwise.} \end{cases}$$

Additionally, for every $i \in \{1, \ldots, UB\}$ and every $e = uv \in E$, consider the binary variable

$$o_{iuv} = \begin{cases} 1, \text{ if edge } e = uv \text{ is in the quasi-clique indexed by } i, \\ 0, \text{ otherwise.} \end{cases}$$

In this way, MQCPP can be formulated as

$$(STDs) \quad \min\sum_{i=1}^{UB} y_i \tag{16}$$

$$(4) - (5), (10) - (12),$$

$$o_{iuv} \le x_{iu}, \qquad \forall i = 1, \dots, UB, \ e = uv \in E, \qquad (17)$$

$$\sum_{v=1}^{n} x_{iv} = \sum_{k=1}^{UB_k} k \cdot z_{ik}, \qquad \forall i = 1, \dots, UB, \qquad (19)$$

$$\sum_{k=1}^{UB_k} z_{ik} = y_i, \qquad \forall \ i = 1, \dots, UB, \qquad (20)$$

$$\sum_{uv\in E} o_{iuv} \ge \gamma \cdot \sum_{k=1}^{UB_k} \frac{k \cdot (k-1)}{2} z_{ik}, \qquad \forall i = 1, \dots, UB, \qquad (21)$$

$$z_{ik} \in \{0, 1\}, \qquad \forall i = 1, \dots, UB, \quad k = 1, \dots, UB_k, \qquad (22)$$

$$o_{iuv} \in \{0, 1\}, \qquad \forall i = 1, \dots, UB, \quad e = uv \in E. \qquad (23)$$

Constraints (17)-(18) guarantee that an edge is in a given quasi-clique only if both its extremities belong to that partition. Constraints (19) force the number of vertices in each quasi-clique to be equal to the value determined by the z variables. Constraints (20) determine that a given quasi-clique is nonempty only if it is used. Constraints (21) ensure the density of every quasi-clique is greater than or equal to γ . Constraints (22)-(23) ensure the integrality requirements on the variables are met.

Formulation by representatives 3.3

e

In this section, we provide a formulation by representatives for MQCPP. Such an approach has been successfully applied in the literature to formulate different partition problems (Campêlo et al., 2005; Frota et al., 2010; Bahiense et al., 2014; Melo and Ribeiro, 2015; Melo et al., 2021b).

Define, for every two vertices $u, v \in V$ with $u \leq v$, the binary variable

$$X_{uu} = \begin{cases} 1, \text{ if vertex } u \text{ is the representative of a quasi-clique belonging to the partition,} \\ 0, \text{ otherwise;} \end{cases}$$

$$X_{uv} = \begin{cases} 1, \text{ if vertex } v \text{ belongs to the quasi-clique represented by vertex } u, \text{ with } u < v, \\ 0, \text{ otherwise.} \end{cases}$$

Furthermore, for any triplet $u, v, v' = 1, \ldots, n : u \le v < v'$, consider the binary variable

$$W_{uvv'} = \begin{cases} 1, \text{ if vertices } v \text{ and } v' \text{ are both in the quasi-clique represented by vertex } u, \\ 0, \text{ otherwise.} \end{cases}$$

A formulation by representatives for MQCPP can be given by:

$$(REP) \quad \min_{u \in V} \sum_{u \in V} X_{uu} \tag{24}$$

$$\sum_{u=1} X_{uv} = 1, \qquad \forall v \in V, \qquad (25)$$

$$X_{uv} \leq X_{uu}, \qquad \forall u, v \in V : u < v, \qquad (26)$$

$$X_{uv} + X_{uv'} \leq W_{uvv'} + 1, \qquad \forall u, v, v' \in V : u \leq v < v', \qquad (27)$$

$$\forall u, v, v' \in V : u \leq v < v', \tag{27}$$

$$W_{uvv'} \le X_{uv}, \qquad \forall \ u, v, v' \in V : u \le v < v', \tag{28}$$

$$W_{uvv'} \le X_{uv'}, \qquad \forall u, v, v' \in V : u \le v < v', \qquad (29)$$

$$\sum_{\substack{v \in V \\ u \le v}} \sum_{\substack{v' \in V \\ v < v', vv' \in E}} W_{uvv'} \ge \gamma \cdot \sum_{\substack{v \in V \\ u \le v}} \sum_{\substack{v' \in V \\ v < v'}} W_{uvv'}, \qquad \forall u = 1, \dots, n-1,$$
(30)

$$X_{uv} \in \{0,1\}, \qquad \forall u, v \in V : u \le v, \qquad (31)$$

$$W_{uvv'} \in \{0, 1\}, \qquad \forall u, v, v' \in V : u \le v < v'.$$
(32)

The objective function (24) minimizes the number of quasi-cliques in the partition. Constraints (25) establish that each vertex belongs to exactly one quasi-clique. Constraints (26) guarantee that a vertex may be represented by vertex v only if v is the representative vertex of a quasi-clique that belongs to the partition. Constraints (27)-(29) enforce that if both vertices v and v' appear in the quasi-clique represented by vertex u, then the existence or not of the edge between them will affect the density of this quasi-clique. Constraints (30) enforce that the density of every quasi-clique should be greater than or equal to γ . Lastly, constraints (31)-(32) ensure the integrality requirements of the variables.

Similarly to what was observed earlier in Section 3.1, if one wishes to avoid equality constraints in formulation REP, constraints (25) can be replaced by

$$\sum_{u=1}^{v} X_{uv} \ge 1, \qquad \forall v \in V, \tag{33}$$

$$X_{uv} + X_{u'v} \le 1, \qquad \forall u, u', v \in V : u < u' \le v.$$

$$(34)$$

3.4 Formulation by representatives using quasi-clique size decomposition

The next formulation combines the ideas of quasi-clique size decompositions and formulations by representatives, already applied in Sections 3.2 and 3.3, respectively.

To formulate the integer program, for every $u \in V$ and $k \in \{1, \ldots, UB_k\}$, consider the binary variable

$$Z_{uk} = \begin{cases} 1, \text{ if a quasi-clique represented by vertex } u \text{ has size } k, \\ 0, \text{ otherwise.} \end{cases}$$

Also, for every $u \in V$ and every $e = vv' \in E$, with $u \leq v < v'$, define the binary variable

$$O_{uvv'} = \begin{cases} 1, \text{ if edge } e = vv' \text{ is in the quasi-clique represented by } u, \\ 0, \text{ otherwise.} \end{cases}$$

Consequently, MQCPP can be formulated as

$$(REPs) \quad \min \sum_{u \in V} X_{uu} \tag{35}$$

$$(25) - (26), (31) - (32), O_{uvv'} \le X_{uv}, \qquad \forall \ u \in V, \ e = vv' \in E : u \le v < v',$$
(36)

$$O_{uvv'} \le X_{uv'}, \qquad \forall \ u \in V, \ e = vv' \in E : u \le v < v', \qquad (37)$$

$$\sum_{v=u}^{n} X_{uv} = \sum_{k=1}^{UB_k} k \cdot Z_{uk}, \qquad \forall u \in V, \qquad (38)$$

$$\sum_{k=1}^{UB_k} Z_{uk} = X_{uu}, \qquad \qquad \forall \ u \in V, \qquad (39)$$

$$\sum_{\substack{e=vv'\in E\\u\leq v$$

$$Z_{uk} \in \{0,1\}, \qquad \forall u \in V, \quad k = 1, \dots, UB_k, \qquad (41)$$

$$O_{uvv'} \in \{0, 1\}, \qquad \forall u \in V, \ e = vv' \in E : u \le v < v'.$$
 (42)

The explanations for constraints (36)-(42) are similar to those for constraints (17)-(23), with the difference that the variables correspond to the representative vertices. Notice that the occurrences of UB_k in formulation REPs can be replaced by min{ $UB_k, n-v+1$ }.

4 Multi-start greedy randomized heuristic

In this section, we describe a multi-start greedy randomized heuristic to provide initial feasible solutions (warm starts) to the proposed formulations. It is based on the HCB constructive heuristic (Pinto et al., 2018) for the maximum quasi-clique problem. HCB is a greedy randomized heuristic based on the potential differences introduced in Abello et al. (1999). The reader is referenced to the authors's original work (Pinto et al., 2018), in which more details can be obtained.

The proposed heuristic is detailed in Algorithm 1, and takes as inputs the graph G, the value of γ , and a parameter α used to define the size of the restricted candidate lists for the HCB constructive heuristic. Line 1 initializes the best known solution with n quasi-cliques, one for each vertex in V. Each iteration of the outer while loop in lines 2-10 builds a new greedy randomized solution. While a stopping condition is not reached, line 3 initializes the set of vertices which were not yet included in the current partition with all the vertices of G. Line 4 sets the current partial solution S as an empty set. Next, each iteration of the inner while loop in lines 5-8 adds a new maximal quasi-clique to the partial solution S. While there are still vertices which are not in the partial solution (i.e., F is not empty), line 6 calls the HCB heuristic to generate a new maximal quasi-clique Q in G[F]. After that, line 7 adds Q to the partial solution S. Line 8 updates the set F by removing the elements in Q. After the inner loop, in lines 9-10, the best solution is updated whenever its cardinality is greater than that of the new constructed quasi-clique partition S. Finally, the best obtained solution S^{best} is returned in line 11.

```
Algorithm 1: Multi-start Greedy Randomized (G, \gamma, \alpha)
  1 S^{\text{best}} \leftarrow \{\{v_1\}, \dots, \{v_n\}\}\};
  \mathbf{2}
     while stopping condition is not met do
           F \leftarrow V;
  3
           S \leftarrow \emptyset;
  4
           while F \neq \emptyset do
  \mathbf{5}
                 Q \leftarrow HCB(G[F], \gamma, \alpha);
  6
                S \leftarrow S \cup \{Q\};
  7
                F \leftarrow F \setminus Q;
  8
                |S| < |S^{best}| then
  9
                S^{\text{best}} \leftarrow S;
10
11 return S^{\text{best}}:
```

5 Computational experiments

This section summarizes the performed computational experiments. All the tests were executed on a machine running under Ubuntu GNU/Linux, with an Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz processor and 16Gb of RAM. The multi-start greedy randomized heuristic was implemented in C++. The integer programming formulations were tackled using the MIP solver Gurobi 9.0.2.

5.1 Benchmark instances

Each benchmark instance corresponds to an input graph and a value for γ . The considered input graphs are displayed in Table 1, sorted by |V|. The graphs correspond to: real networks that are commonly used in the literature, which were recently used for the longest induced path problem and referenced in Matsypura et al. (2019); instances from the DIMACS Implementation Challenges (DIMACS, 2021); and instances from the Moviegalaxies data set (Kaminski et al., 2018) (https://doi.org/10.7910/DVN/T4HBA3), which represent social connectivity of the characters in movies. The table shows the input graphs, followed by their numbers of vertices and edges, as well as their densities. For each of the 23 input graphs, instances were considered for every $\gamma \in \{0.999, 0.950, 0.900, 0.800, 0.700, 0.600, 0.500, 0.400, 0.300\}$, leading to a total of 207 instances. We remark that we selected these graphs based on the fact that they are widely used in the literature and also considering the limitations of the integer programming approaches, which have $O(|V|^3)$ variables each.

Table 1: Input graphs used as benchmarks.

~	no in impac grapia			
	Input graph	V	E	d(G)
	Memento	14	19	0.2088
	The_X_Files	24	41	0.1486
	Alien_3	25	77	0.2567
	high-tech	33	91	0.1723
	karate	34	78	0.1390
	mexican	35	117	0.1966
	sawmill	36	62	0.0984
	tailorS1	39	158	0.2132
	chesapeake	39	170	0.2294
	Batman_Returns	51	124	0.0973
	attiro	59	128	0.0748
	krebs	62	153	0.0809
	dolphins	62	159	0.0841
	prison	67	142	0.0642
	sanjuansur	75	144	0.0519
	jean	77	254	0.0868
	3-FullIns_3	80	346	0.1095
	david	87	406	0.1085
	myciel6	95	755	0.1691
	4-FullIns_3	114	541	0.0840
	ieeebus	118	179	0.0259
	sfi	118	200	0.0290
	anna	138	493	0.0522
-				

5.2 Tested approaches and settings

We compare the standard formulation (STD), the formulation using quasi-clique size decompositions (STDs), the formulation by representatives (REP), and the formulation by representatives using quasi-clique size decomposition (REPs), all of them presented in Section 3. The solutions obtained by the multi-start greedy randomized heuristic (MSH), described in Section 4, were provided as initial feasible solutions for each execution of all the formulations.

We defined a time limit of 30 seconds as the stopping criterion for obtaining an initial feasible solution with the heuristic MSH. Similarly to Pinto et al. (2018), we set $\alpha = 0.09$. For solving the formulations, Gurobi was executed with the standard configurations using a single thread and a time limit of 3600 seconds was imposed on each run.

5.3 Computational results

This section summarizes the computational results. Individualized results for each instance are available in Appendix A. Table 2 summarizes the results obtained by the formulations, assembled by the input graphs. The first column identifies the input graph, followed by the number of instances (i.e., the number of values

for γ). All reported averages in other columns of the table are taken over the nine instances with different values of γ for the specified input graph. The third column gives the average best solution value obtained by MSH within 30 seconds. Next, for each formulation, the table provides the average best solution value, the number of instances solved to optimality, the average time in seconds for solving those that were optimally solved, and the average gap in percentage for the instances that were not solved to optimality within 3600 seconds. In each line, the smallest average best solution value (column best) and the largest number of instances solved to optimality (column #opt) are shown in boldface. The results show that formulation REPv clearly outperforms the other formulations when it comes to the average best solution values, while REP is slightly better than REPv when it comes to the number of instances solved to optimality. Besides, it can be seen that the heuristic MSH achieved, in low computational times, solutions whose best values were, in several cases, not far from the smallest ones reported in the table. Additionally, STD was the formulation that encountered most difficulties in improving the solutions obtained by MSH. It can also be observed from this table that REPv obtains lower average open gaps for most of the input graphs.

Table 3 summarizes the results obtained by the formulations, categorized by the values of γ . The first column shows each considered value of γ , followed by the number of instances (i.e., the number of input graphs). The remaining columns are the same as those in Table 2, except for the fact that all reported averages are taken over the 23 instances corresponding to the different input graphs, for each specific value of γ . It is noticeable from this table that the two formulations by representatives (REP and REPv) dominate the others when it comes to the average best solution values, with REP outperforming all the others for larger values of γ and REPv showing the best performance for medium and lower values of γ . Regarding the number of instances solved to optimality, REP performed the best for those with $\gamma \geq 0.800$. For $\gamma \leq 0.700$, STDv and REPv achieved more optimal solutions for three values of gamma each. It should be noticed that STDv achieved the largest number of instances solved to optimality for the three smallest values of γ .

Figure 4 shows the fraction in percent of the instances solved to optimality by at least one of the formulations. The plot indicates that, considering the input graphs used in the benchmark, the instances with large values of γ (0.999, 0.950, and 0.900) tend to be easier to solve, while those with intermediary values (0.700 and 0.600) appear to be the most difficult ones. Furthermore, the plot shows that, together, the proposed approaches could solve nearly 59% of the instances to optimality.

Table 2: Summary of the results obtained by the four formulations displayed by the input graphs.																		
MSH STD					STDv					RI	EP		REPv					
Input graph	#inst	best	best	#opt	time	$_{\mathrm{gap}}$	best	#opt	time	$_{\mathrm{gap}}$	best	# opt	time	$_{\mathrm{gap}}$	best	#opt	time	$_{\mathrm{gap}}$
					(s)	(%)			(s)	(%)			(s)	(%)			(s)	(%)
Memento	9	8.4	7.8	9	10.6		7.8	9	1.1		7.8	9	0.5		7.8	9	0.1	
The_X_Files	9	10.0	9.8	3	11.4	35.1	9.8	3	0.7	19.8	9.8	9	53.5		9.8	9	3.9	
Alien_3	9	7.0	6.6	5	151.4	29.3	6.6	5	81.2	28.7	6.6	9	173.5		6.6	9	32.4	
high-tech	9	10.6	10.4	2	76.2	51.4	10.4	3	62.9	45.3	10.4	6	447.6	27.8	10.4	9	572.4	
karate	9	13.8	13.1	2	6.8	58.5	13.0	3	24.8	42.3	12.9	7	434.2	41.4	12.9	9	32.5	
mexican	9	8.1	8.1	3	12.3	47.1	8.0	4	179.5	39.8	8.1	6	112.9	36.0	8.1	6	1320.6	20.4
sawmill	9	12.8	12.8	1	1.3	57.5	12.8	3	33.9	36.7	12.8	6	54.7	23.6	12.8	9	45.9	
tailorS1	9	10.3	10.2	3	283.0	62.8	10.1	2	0.5	49.9	9.8	6	479.7	49.0	9.9	2	20.8	47.5
chesapeake	9	10.8	10.6	3	749.5	61.5	10.4	3	25.1	53.0	10.3	4	99.5	45.5	10.3	3	871.4	34.6
Batman_Returns	9	14.3	14.3	1	11.5	69.4	13.6	2	33.2	39.2	13.7	4	260.8	52.8	13.3	7	1477.5	28.4
attiro	9	18.8	18.8	1	14.6	76.6	18.6	1	805.3	41.1	18.8	4	367.4	51.4	18.6	5	729.2	20.7
krebs	9	23.0	22.8	1	227.5	80.1	21.7	1	392.8	51.3	22.0	3	129.3	55.2	21.3	1	1625.7	47.9
dolphins	9	20.3	20.3	1	7.5	79.4	19.3	1	1277.3	50.8	19.8	3	32.3	53.6	19.2	0		34.4
prison	9	18.9	18.9	1	6.0	77.3	18.7	0		34.7	18.8	4	407.9	55.8	18.6	3	1916.2	23.2
sanjuansur	9	24.7	24.7	1	32.2	82.7	24.6	0		39.9	24.3	4	435.3	55.9	24.2	5	1244.1	21.6
jean	9	26.8	26.3	1	17.5	85.0	24.1	1	213.1	57.6	25.9	1	1.7	63.9	23.0	1	323.3	53.3
3-FullIns_3	9	26.0	25.9	1	38.3	81.4	25.9	0		65.5	25.4	2	1277.4	64.3	26.0	0		69.8
david	9	27.7	27.2	1	188.9	87.0	24.6	0		59.7	26.6	2	1501.1	76.3	22.8	0		58.8
myciel6	9	39.6	38.9	0		81.3	38.2	0		83.5	36.9	1	2.5	89.3	35.3	0		82.0
4-FullIns_3	9	38.7	38.6	0		79.8	38.6	0		75.2	38.0	1	7.1	74.0	38.4	0		74.2
ieeebus	9	45.9	45.4	0		82.9	45.4	0		43.8	45.2	3	218.3	64.1	44.3	5	2377.4	36.2
sfi	9	53.4	53.4	0		83.6	50.6	0		54.3	51.7	1	5.1	58.8	47.8	3	1652.4	37.9
anna	9	67.7	67.7	1	264.5	95.3	65.8	0		80.7	65.7	1	9.8	76.4	62.3	0		77.9
Average		23.4	23.2		111.1	70.2	22.5		223.7	49.7	22.7		283.1	55.7	21.9		838.0	45.2
Total	207			41				41				96				95		

Table 2: Summary of the results obtained by the four formulations displayed by the input graphs.

Table 3: Summary of the results obtained by the four formulations displayed by the values of γ .

Table 3: Summary of the results obtained by the four formulations displayed by the values of γ .																		
MSH STD						STDv					RE	ΞP		REPv				
γ	#inst	best	best	#opt	time	gap	best	#opt	time	$_{\rm gap}$	best	#opt	time	gap	best	#opt	time	$_{\rm gap}$
					(s)	(%)			(s)	(%)			(s)	(%)			(s)	(%)
0.999	23	31.5	31.2	19	42.8	6.7	31.5	1	1.1	53.6	31.2	23	1.7		31.5	14	883.7	59.7
0.950	23	31.4	31.4	1	0.1	77.8	31.4	1	1.7	55.5	31.1	18	335.1	27.6	31.3	13	648.7	61.7
0.900	23	30.8	30.8	1	13.2	79.8	30.8	1	1.3	55.8	30.3	16	101.7	33.9	30.8	13	1023.4	63.1
0.800	23	28.9	28.7	1	14.1	77.9	28.9	1	1.2	56.0	28.2	12	579.0	48.3	28.6	10	831.1	55.8
0.700	23	27.4	27.1	1	19.9	75.6	27.1	2	194.8	58.0	26.8	6	399.3	56.5	26.7	8	953.1	55.6
0.600	23	19.7	19.5	2	380.3	72.4	19.0	3	228.6	44.7	18.9	5	742.9	67.2	18.0	7	598.2	48.0
0.500	23	16.7	16.5	2	16.7	69.0	15.4	8	60.3	48.5	15.7	3	113.9	68.2	13.7	7	489.5	42.2
0.400	23	13.7	13.4	6	528.8	69.9	11.3	10	7.7	41.1	12.8	5	505.7	68.6	10.1	10	38.3	42.0
0.300	23	10.1	9.7	8	21.1	68.1	7.3	14	192.3	36.4	8.9	8	129.5	69.4	6.5	13	267.2	39.2



Figure 4: Fraction in percent of the instances solved to optimality by at least one formulation within the time of 3600 seconds.

6 Concluding remarks

In this paper, we considered the minimum quasi-clique partitioning problem (MQCPP), which has applicability in clustering and network analysis. We formally introduced the problem and showed that its decision version is NP-complete even when one asks for a partition into two γ -cliques, i.e., a bipartition. This interesting result contrasts with that for the minimum clique partitioning problem, whose decision version is NP-complete for obtaining a partition into more than two cliques, but polynomially solvable for two.

Furthermore, we proposed four integer programming formulations and a multi-start greedy randomized heuristic for MQCPP. The computational experiments showed that the formulation by representatives (REP) and the formulation by representatives using the idea of clique-size decomposition (REPs) are those with the best performances. While REP achieved a larger number of instances solved to optimality, REPs outperformed the others when it comes to the best obtained solution values. Besides, the results also indicate that instances with intermediate values of γ tend to be more difficult to be solved by the proposed formulations. Furthermore, the experiments showed that the proposed heuristic can obtain, in low computational times, initial feasible solutions of reasonable quality to be offered to the formulations.
Acknowledgments

Work of Rafael A. Melo was supported by the State of Bahia Research Foundation (FAPESB) and by the Brazilian National Council for Scientific and Technological Development (CNPq). Part of this work was developed while the author was a visiting scholar at the Department of Computer Science of Universidade Federal de Minas Gerais, Brazil. Work of Celso C. Ribeiro was partially supported by CNPq research grants 309869/2020-0 and 425778/2016-9, and by FAPERJ research grant E-26/200.926/2021. Work of Jose A. Riveaux was sponsored by a CNPq scholarship. This work was also was partially sponsored by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), under Finance Code 001.

References

- Abello, J., Pardalos, P.M., Resende, M.G.C., 1999. On maximum clique problems in very large graphs. In Abello, J.M. and Vitter, J.S. (eds), *External Memory Algorithms*, American Mathematical Society, pp. 119–130.
- Abello, J., Resende, M., Sudarsky, S., 2002. Massive quasi-clique detection. In Abello, J. and Vitter, J. (eds), Proceedings of the 5th Latin American Symposium on the Theory of Informatics, Lecture Notes in Computer Science. Vol. 2286. Springer, Berlin, pp. 598–612.
- Bahiense, L., Frota, Y.A.M., Noronha, T.F., Ribeiro, C.C., 2014. A branch-and-cut algorithm for the equitable coloring problem using a formulation by representatives. *Discrete Applied Mathematics* 164, 34–46.
- Basu, S., Sengupta, D., Maulik, U., Bandyopadhyay, S., 2014. A strong nash stability based approach to minimum quasi clique partitioning. In 2014 Sixth International Conference on Communication Systems and Networks, IEEE, Bangalore, pp. 1–6.
- Blum, C., Djukanovic, M., Santini, A., Jiang, H., Li, C.M., Manyà, F., Raidl, G.R., 2021. Solving longest common subsequence problems via a transformation to the maximum clique problem. *Computers & Operations Research* 125, 105089.
- Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M., 1999. The maximum clique problem. In Pardalos, P.M., Du, D.Z. and Graham, R.L. (eds), Handbook of Combinatorial Optimization. Springer, pp. 1–74.
- Brimberg, J., Janićijević, S., Mladenović, N., Urošević, D., 2017. Solving the clique partitioning problem as a maximally diverse grouping problem. *Optimization Letters* 11, 1123–1135.
- Campêlo, M., Campos, V., Corrêa, R., 2005. On the asymmetric representatives formulation for the vertex coloring problem. *Electronic Notes in Discrete Mathematics* 19, 337–343.
- Campêlo, M., Corrêa, R., Frota, Y., 2004. Cliques, holes and the vertex coloring polytope. Information Processing Letters 89, 159–164.
- De Amorim, S.G., Barthélemy, J.P., Ribeiro, C.C., 1992. Clustering and clique partitioning: Simulated annealing and tabu search approaches. *Journal of Classification* 9, 17–41.
- Dell'Amico, M., Montemanni, R., Novellani, S., 2022. Exact models for the flying sidekick traveling salesman problem. International Transactions in Operational Research 29, 1360–1393.
- DIMACS, 2021. Implementation challenges. Online reference at http://dimacs.rutgers.edu/Challenges/ last visited on November 27, 2021.
- Frota, Y., Maculan, N., Noronha, T.F., Ribeiro, C.C., 2010. A branch-and-cut algorithm for partition coloring. Networks: An International Journal 55, 194–204.

- Garey, M.R., Johnson, D.S., Stockmeyer, L., 1974. Some simplified NP-complete problems. In *Proceedings* of the Sixth Annual ACM Symposium on Theory of Computing, ACM, Seattle, pp. 47–63.
- Glaria, F., Hernández, C., Ladra, S., Navarro, G., Salinas, L., 2021. Compact structure for sparse undirected graphs based on a clique graph partition. *Information Sciences* 544, 485–499.
- Gramm, J., Guo, J., Hüffner, F., Niedermeier, R., 2009. Data reduction and exact algorithms for clique cover. ACM Journal of Experimental Algorithmics 13, 2.2–2.15.
- Grötschel, M., Wakabayashi, Y., 1990. Facets of the clique partitioning polytope. *Mathematical Programming* 47, 367–387.
- Hu, H., Yan, X., Huang, Y., Han, J., Zhou, X.J., 2005. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics* 21, i213–i221.
- Jensen, T.R., Toft, B., 2011. Graph Coloring Problems, Vol. 39. Wiley.
- Kaminski, J., Schober, M., Albaladejo, R., Zastupailo, O., Hidalgo, C., 2018. Moviegalaxies Social networks in movies. Online reference at https://doi.org/10.7910/DVN/T4HBA3 last visited on November 27, 2021.
- Karp, R.M., 1972. Reducibility among combinatorial problems. In Miller, R.E. and Thatcher, J.W. (eds), Complexity of Computer Computations, Plenum Press, pp. 85—103.
- Lee, V.E., Ruan, N., Jin, R., Aggarwal, C., 2010. A survey of algorithms for dense subgraph discovery. In Aggarwal, C. and Wang, H. (eds), *Managing and Mining Graph Data*, *Advances in Database Systems*. Vol. 40. Springer, pp. 303–336.
- Li, C.M., Jiang, H., Manyà, F., 2017. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & Operations Research* 84, 1–15.
- Malaguti, E., Toth, P., 2010. A survey on vertex coloring problems. International Transactions in Operational Research 17, 1–34.
- Marinelli, F., Pizzuti, A., Rossi, F., 2021. LP-based dual bounds for the maximum quasi-clique problem. Discrete Applied Mathematics 296, 118–140.
- Marzo, R.G., Melo, R.A., Ribeiro, C.C., Santos, M.C., 2022. New formulations and branch-and-cut procedures for the longest induced path problem. *Computers & Operations Research* 139, 105627.
- Matsypura, D., Veremyev, A., Prokopyev, O.A., Pasiliao, E.L., 2019. On exact solution approaches for the longest induced path problem. *European Journal of Operational Research* 278, 546–562.
- Melo, R.A., Queiroz, M.F., Ribeiro, C.C., 2021a. Compact formulations and an iterated local search-based matheuristic for the minimum weighted feedback vertex set problem. *European Journal of Operational Research* 289, 75–92.
- Melo, R.A., Queiroz, M.F., Santos, M.C., 2021b. A matheuristic approach for the b-coloring problem using integer programming and a multi-start multi-greedy randomized metaheuristic. *European Journal* of Operational Research 295, 66–81.
- Melo, R.A., Ribeiro, C.C., 2015. Improved solutions for the freight consolidation and containerization problem using aggregation and symmetry breaking. *Computers & Industrial Engineering* 85, 402–413.
- Méndez-Díaz, I., Zabala, P., 2006. A branch-and-cut algorithm for graph coloring. Discrete Applied Mathematics 154, 826–847.
- Oosten, M., Rutten, J.H., Spieksma, F.C., 2001. The clique partitioning problem: facets and patching facets. Networks: An International Journal 38, 209–226.

- Östergård, P.R., 2002. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics* 120, 197–207.
- Pardalos, P.M., Xue, J., 1994. The maximum clique problem. Journal of Global Optimization 4, 301–328.
- Pattillo, J., Veremyev, A., Butenko, S., Boginski, V., 2013. On the maximum quasi-clique problem. Discrete Applied Mathematics 161, 244–257.
- Pattillo, J., Youssef, N., Butenko, S., 2013. On clique relaxation models in network analysis. European Journal of Operational Research 226, 9–18.
- Pinto, B.Q., Ribeiro, C.C., Riveaux, J.A., Rosseti, I., 2021. A BRKGA-based matheuristic for the maximum quasi-clique problem with an exact local search strategy. *RAIRO: Recherche Opérationnelle* 55, S741 S763.
- Pinto, B.Q., Ribeiro, C.C., Rosseti, I., Plastino, A., 2018. A biased random-key genetic algorithm for the maximum quasi-clique problem. *European Journal of Operational Research* 271, 849–865.
- Ribeiro, C.C., Riveaux, J., 2019. An exact algorithm for the maximum quasi-clique problem. *International Transactions in Operational Research* 26, 2199–2229.
- Sanei-Mehri, S.V., Das, A., Hashemi, H., Tirthapura, S., 2021. Mining largest maximal quasi-cliques. ACM Transactions on Knowledge Discovery from Data 15, 1–21.
- Spirin, V., Mirny, L.A., 2003. Protein complexes and functional modules in molecular networks. Proceedings of the National Academy of Sciences 100, 12123–12128.
- Tsourakakis, C., Bonchi, F., Gionis, A., Gullo, F., Tsiarli, M., 2013. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *Proceedings of the 19th ACM SIGKDD In*ternational Conference on Knowledge Discovery and Data Mining, Chicago, pp. 104–112.
- Veremyev, A., Prokopyev, O.A., Butenko, S., Pasiliao, E.L., 2016. Exact MIP-based approaches for finding maximum quasi-clique and dense subgraphs. Computational Optimization and Applications 64, 177–214.
- Verma, A., Butenko, S., 2013. Network clustering via clique relaxations: A community based approach. In Bader, D.A., Meyerhenke, H., Sanders, P. and Wagner, D. (eds), Graph Partitioning and Graph Clustering, Contemporary Mathematics. Vol. 588. American Mathematical Society, pp. 129–139.
- Verteletskyi, V., Yen, T.C., Izmaylov, A.F., 2020. Measurement optimization in the variational quantum eigensolver using a minimum clique cover. *The Journal of Chemical Physics* 152, 124114.
- Wu, Q., Hao, J.K., 2015. A review on algorithms for maximum clique problems. European Journal of Operational Research 242, 693–709.
- Zhou, Q., Benlic, U., Wu, Q., 2020. An opposition-based memetic algorithm for the maximum quasi-clique problem. European Journal of Operational Research 286, 63–83.

Appendix A Detailed results

Table 4 presents the results obtained by each of the formulations for each instance. The first two columns indicate the input graph and the value of γ . The third column gives the value of the best solution obtained by the heuristic MSH within 30 seconds. The next columns show, for each of the formulations (STD, STDv, REP, and REPv), the value of the best obtained solution, the best bound achieved by the solver, and the running time in seconds.

		MSH		STD			STDv			REP			REPv	
Input graph	γ	best	best	bbound	time (s)	best	bbound	time (s)	best	bbound	time (s)	best	bbound	time (s)
Memento	0.999	10	10	10	0.024	10	10	1.123	10	10	0.004	10	10	0.103
Memento	0.950	10	10	10	0.050	10	10	1.747	10	10	0.004	10	10	0.097
Memento	0.900	10	10	10	13.215	10	10	1.260	10	10	0.009	10	10	0.130
Memento	0.800	9	9	9	14.072	9	9	1.163	9	9	0.049	9	9	0.108
Memento	0.700	9	9	9	19.878	9	9	1.467	9	9	0.112	9	9	0.137
Memento	0.600	8	8	8	24.206	8	8	1.984	8	8	0.332	8	8	0.118
Memento	0.500	8	6	6	15.974	6	6	0.650	6	6	0.468	6	6	0.141
Memento	0.400	7	5	5	7.290	5	5	0.641	5	5	1.144	5	5	0.261
Memento	0.300	5	3	3	0.710	3	3	0.108	3	3	2.197	3	3	0.180
The_X_Files	0.999	14	14	14	0.095	14	12	3600.004	14	14	0.016	14	14	1.454
The_X_Files	0.950	14	14	9	3600.006	14	11	3600.004	14	14	0.040	14	14	2.320
The_X_Files	0.900	14	14	7	3600.005	14	11	3600.004	14	14	0.147	14	14	2.918
The_X_Files	0.800	13	13	8	3600.004	13	10	3600.005	13	13	0.802	13	13	3.622
The_X_Files	0.700	12	11	7	3600.005	11	8	3600.004	11	11	2.202	11	11	7.873
The_X_Files	0.600	9	9	6	3600.005	9	8	3600.004	9	9	49.561	9	9	9.815
The_X_Files	0.500	6	6	5	3600.004	6	6	1.897	6	6	110.959	6	6	4.863
The_X_Files	0.400	5	4	4	31.956	4	4	0.274	4	4	300.454	4	4	2.397
The_X_Files	0.300	3	3	3	2.106	3	3	0.050	3	3	17.577	3	3	0.065
Alien_3	0.999	11	11	11	0.121	11	7	3600.004	11	11	0.033	11	11	35.667
Alien_3	0.950	11	11	8	3600.005	11	7	3600.004	11	11	0.164	11	11	64.696
Alien_3	0.900	10	10	6	3600.005	10	8	3600.004	10	10	1.172	10	10	46.149
Alien_3	0.800	9	9	6	3600.006	9	7	3600.005	9	9	68.530	9	9	72.105
Alien_3	0.700	8	6	5	3600.005	6	6	388.034	6	6	106.320	6	6	36.423
Alien_3	0.600	5	5	5	736.402	5	5	17.340	5	5	1069.271	5	5	31.611
Alien_3	0.500	4	3	3	17.415	3	3	0.259	3	3	230.227	3	3	4.113
Alien_3	0.400	3	2	2	3.255	2	2	0.192	2	2	85.667	2	2	0.402
Alien_3	0.300	2	2	2	0.052	2	2	0.020	2	2	0.107	2	2	0.035
high-tech	0.999	16	16	16	0.256	16	9	3600.009	16	16	0.072	16	16	103.647
high-tech	0.950	16	16	6	3600.109	16	8	3600.010	16	16	0.810	16	16	180.802
high-tech	0.900	15	15	5	3600.003	15	7	3600.010	15	15	3.426	15	15	326.564
high-tech	0.800	14	14	5	3600.002	14	7	3600.010	14	14	111.734	14	14	558.308
high-tech	0.700	12	12	5	3600.002	12	6	3600.010	12	12	2120.085	12	12	1163.487
high-tech	0.600	9	8	4	3600.191	8	6	3600.011	8	6	3600.008	8	8	827.532
high-tech	0.500	6	6	4	3600.011	6	6	187.855	6	4	3600.031	6	6	1981.515
high-tech	0.400	4	4	3	3600.011	4	4	0.711	4	3	3600.008	4	4	7.156
high-tech	0.300	3	3	3	152.091	3	3	0.242	3	3	449.544	3	3	2.866
												С	ontinued on	next page

Table 4: Detailed results obtained by the four formulations for the minimum quasi-clique partitioning problem.

Table 4 – continued from previous page														
		MSH		STD			STDv		Ĭ	REP			REPv	
Input graph	γ	best	best	bbound	time (s)	best	bbound	time (s)	best	bbound	time (s)	best	bbound	time (s)
karate	0.999	20	20	20	0.214	20	11	3600.010	20	20	0.061	20	20	18.524
karate	0.950	20	20	6	3600.012	20	11	3600.010	20	20	0.409	20	20	25.864
karate	0.900	19	19	6	3600.005	19	10	3600.011	19	19	4.338	19	19	23.378
karate	0.800	18	17	5	3600.008	18	9	3600.011	17	17	26.329	17	17	52.001
karate	0.700	17	15	5	3600.007	15	8	3600.011	15	15	138.245	15	15	69.826
karate	0.600	13	11	4	3600.002	10	8	3600.011	10	10	2304.545	10	10	55.491
karate	0.500	9	8	4	3600.002	7	7	67.598	7	4	3600.088	7	7	40.334
karate	0.400	5	5	4	3600.012	5	5	6.612	5	3	3600.236	5	5	6.603
karate	0.300	3	3	3	13.399	3	3	0.067	3	3	565.466	3	3	0.236
mexican	0.999	13	13	13	0.336	13	7	3600.012	13	13	0.092	13	13	1135.882
mexican	0.950	13	13	6	3600.003	13	7	3600.012	13	13	2.147	13	13	1173.879
mexican	0.900	12	12	5	3600.004	12	7	3600.012	12	12	11.767	12	12	2294.478
mexican	0.800	10	10	5	3600.007	10	6	3600.012	10	10	383.898	10	10	3319.170
mexican	0.700	8	8	5	3600.018	8	6	3600.013	8	6	3600.015	8	7	3600.007
mexican	0.600	7	7	4	3600.015	6	6	666.593	7	4	3600.002	7	5	3600.007
mexican	0.500	5	5	3	3600.014	5	5	51.217	5	3	3600.028	5	4	3600.008
mexican	0.400	3	3	3	36.522	3	3	0.134	3	3	279.281	3	3	0.332
mexican	0.300	2	2	2	0.102	2	2	0.048	2	2	0.151	2	2	0.076
sawmill	0.999	18	18	18	1.308	18	12	3600.013	18	18	0.077	18	18	14.403
sawmill	0.950	18	18	5	3600.028	18	12	3600.014	18	18	0.239	18	18	15.805
sawmill	0.900	18	18	5	3600.020	18	11	3600.013	18	18	1.492	18	18	30.594
sawmill	0.800	16	16	5	3600.003	16	10	3600.013	16	16	6.127	16	16	26.443
sawmill	0.700	16	16	4	3600.018	16	8	3600.013	16	16	28.970	16	16	43.042
sawmill	0.600	11	11	4	3600.002	11	8	3600.014	11	11	290.999	11	11	146.263
sawmill	0.500	8	8	4	3600.003	8	8	98.927	8	7	3600.005	8	8	96.526
sawmill	0.400	6	6	4	3600.015	6	6	2.348	6	4	3600.002	6	6	39.373
sawmill	0.300	4	4	3	3600.003	4	4	0.339	4	3	3600.002	4	4	0.286
tailorS1	0.999	17	17	17	2.667	17	7	3600.017	17	17	0.170	17	8	3600.010
tailorS1	0.950	17	17	5	3600.117	17	6	3600.017	17	17	6.537	17	7	3600.010
tailorS1	0.900	15	15	4	3600.020	15	6	3600.017	15	15	38.344	15	7	3600.010
tailorS1	0.800	13	13	4	3600.003	13	6	3600.017	12	12	971.212	12	6	3600.010
tailorS1	0.700	11	11	4	3600.076	11	5	3600.017	10	5	3600.003	11	5	3600.011
tailorS1	0.600	8	8	4	3600.004	8	5	3600.017	7	3	3600.002	7	4	3600.010
tailorS1	0.500	6	6	3	3600.005	5	4	3600.016	5	3	3600.002	5	4	3600.010
tailorS1	0.400	4	3	3	846.108	3	3	0.923	3	3	1861.806	3	3	41.549
tailorS1	0.300	2	2	2	0.140	2	2	0.066	2	2	0.204	2	2	0.108
chesapeake	0.999	17	17	17	0.516	17	7	3600.017	17	17	0.142	17	17	2603.069
chesapeake	0.950	17	17	5	3600.003	17	6	3600.017	17	17	77.271	17	12	3600.011
chesapeake	0.900	17	17	5	3600.003	17	6	3600.016	16	16	320.062	17	10	3600.010
chesapeake	0.800	15	14	4	3600.004	14	6	3600.016	13	8	3600.002	13	8	3600.010
chesapeake	0.700	12	12	4	3600.003	12	5	3600.017	12	4	3600.002	12	6	3600.010
chesapeake	0.600	8	8	4	3600.071	7	6	3600.018	7	3	3600.009	7	5	3600.011
chesapeake	0.500	5	5	3	3600.003	5	5	74.254	5	3	3600.003	5	4	3600.010
chesapeake	0.400	4	3	3	2247.916	3	3	0.874	4	3	3600.002	3	3	11.122
chesapeake	0.300	2	2	2	0.142	2	2	0.069	2	2	0.362	2	2	0.100
												C	ontinued on	nort noro

Table 4 – continued from previous page														
		MSH		STD			STDv	1	REP			REPv		
Input graph	γ	best	best	bbound	time (s)									
Batman_Returns	0.999	20	20	20	11.467	20	12	3600.005	20	20	0.251	20	20	1177.708
Batman_Returns	0.950	20	20	5	3600.074	20	12	3600.039	20	20	8.742	20	20	1479.540
Batman_Returns	0.900	20	20	4	3600.011	20	10	3600.038	20	20	49.210	20	20	2992.159
Batman_Returns	0.800	17	17	4	3600.008	17	10	3600.038	17	17	985.040	17	13	3600.024
Batman_Returns	0.700	16	16	4	3600.007	16	8	3600.038	15	9	3600.017	15	10	3600.023
Batman_Returns	0.600	11	11	4	3600.007	10	8	3600.039	10	4	3600.005	10	10	3116.494
Batman_Returns	0.500	11	11	3	3600.007	9	6	3600.040	9	3	3600.038	8	8	1299.324
Batman_Returns	0.400	8	8	3	3600.008	6	6	63.842	7	3	3600.005	6	6	273.627
Batman_Returns	0.300	6	6	3	3600.013	4	4	2.650	5	3	3600.005	4	4	3.402
attiro	0.999	27	27	27	14.635	27	15	3600.008	27	27	0.423	27	27	608.158
attiro	0.950	27	27	4	3613.708	27	13	3600.008	27	27	16.097	27	27	452.162
attiro	0.900	27	27	4	3600.012	27	14	3600.008	27	27	41.534	27	27	375.218
attiro	0.800	24	24	4	3614.460	24	11	3600.060	24	24	1411.460	24	24	693.967
attiro	0.700	21	21	3	3602.443	21	11	3600.060	21	17	3600.007	21	18	3600.039
attiro	0.600	14	14	4	3600.011	14	9	3600.062	14	9	3600.024	14	11	3600.036
attiro	0.500	12	12	3	3600.011	12	9	3600.062	12	3	3600.056	12	9	3600.038
attiro	0.400	10	10	3	3600.011	9	7	3600.062	10	3	3600.010	9	7	3600.038
attiro	0.300	7	7	3	3600.012	6	6	805,260	7	3	3600.007	6	6	1516.723
krebs	0.999	33	33	33	227.467	33	12	3600.010	33	33	0.672	33	21	3600.006
krebs	0.950	33	33	5	3600.015	33	13	3600.009	33	33	54.927	33	15	3600.008
krebs	0.900	31	31	4	3600.847	31	12	3600.011	31	31	332.220	31	14	3600.006
krebs	0.800	27	27	4	3600.016	27	11	3600.009	26	24	3600.088	27	12	3601.110
krebs	0.700	25	25	4	3600.031	25	10	3600.010	25	13	3600.008	25	10	3600.557
krebs	0.600	20	19	4	3604.280	20	9	3600.067	18	5	3600.010	18	8	3600,006
krebs	0.500	17	16	3	3600.207	13	8	3600.072	14	3	3600.009	12	7	3600.006
krebs	0.400	13	13	3	3600.013	8	7	3600.064	12	3	3600.013	8	6	3600.006
krebs	0.300	8	8	3	3600.014	5	5	392.798	6	3	3600.013	5	5	1625.738
dolphins	0.999	28	28	28	7.540	28	12	3600.010	28	28	0.433	28	23	3600.040
dolphins	0.950	28	28	4	3676.140	28	12	3600.010	28	28	29.184	28	18	3600.213
dolphins	0.900	27	27	4	3600.035	27	11	3600.010	27	27	67.411	27	17	3600.616
dolphins	0.800	25	25	4	3610.317	25	10	3600.010	25	23	3600.023	25	13	3600.007
dolphins	0.700	23	23	3	3601.896	23	10	3600.011	23	14	3600.025	23	11	3600.008
dolphins	0.600	18	18	4	3600.014	18	9	3600.009	16	5	3600.008	16	9	3600.006
dolphins	0.500	15	15	3	3600.122	12	7	3600.192	14	3	3600.011	12	8	3600.006
dolphins	0.400	11	11	3	3600.014	8	6	3600.073	10	3	3600.017	8	6	3600.006
dolphins	0.300	8	8	3	3600.014	5	5	1277.279	7	3	3600.011	6	5	3600.047
prison	0.999	26	26	26	5.970	26	15	3600.012	26	26	0.664	26	26	2952.480
prison	0.950	26	26	4	3601.860	26	14	3600.016	26	26	41.766	26	26	1091.526
prison	0.900	25	25	4	3600.020	25	14	3600.067	25	25	79.549	25	25	1704.530
prison	0.800	24	24	4	3600.021	24	12	3600.014	24	24	1509.422	24	21	3600.058
prison	0.700	22	22	4	3609.392	22	12	3600.012	22	17	3600.010	22	13	3600.008
prison	0.600	16	16	4	3600.084	15	11	3600.014	15	8	3600.011	15	11	3600.008
prison	0.500	13	13	3	3600.019	13	10	3600.091	13	3	3600.011	12	9	3600.007
prison	0.400	10	10	3	3600.017	10	8	3600.092	10	3	3600.011	10	8	3600.012
prison	0.300	8	8	3	3600.595	7	6	3600.087	8	3	3600.012	7	6	3600.007
*				-							-	0		

					Table 4	4 – cont	inued from	previous pa	ge					
		MSH		STD			STDv			REP			REPv	
Input graph	γ	best	best	bbound	time (s)	best	bbound	time (s)	best	bbound	time (s)	best	bbound	time (s)
sanjuansur	0.999	35	35	35	32.190	35	18	3600.018	35	35	1.059	35	35	837.826
sanjuansur	0.950	35	35	4	3600.032	35	18	3600.018	35	35	89.764	35	35	631.064
sanjuansur	0.900	35	35	4	3600.034	35	17	3600.017	35	35	176.950	35	35	740.888
sanjuansur	0.800	31	31	3	3600.027	31	15	3600.020	31	31	1473.526	31	31	1076.276
sanjuansur	0.700	29	29	4	3600.028	29	13	3600.017	29	20	3600.015	29	29	2934.521
sanjuansur	0.600	20	20	3	3605.590	20	13	3600.018	18	12	3600.018	18	14	3600.012
sanjuansur	0.500	16	16	3	3600.029	16	11	3600.130	15	4	3600.018	15	11	3600.011
sanjuansur	0.400	12	12	3	3600.029	12	9	3600.128	12	3	3600.014	12	9	3600.011
sanjuansur	0.300	9	9	3	3600.185	8	7	3600.119	9	3	3600.015	8	7	3600.544
jean	0.999	35	35	35	17.472	35	12	3600.020	35	35	1.709	35	16	3600.011
jean	0.950	35	35	4	3600.031	35	12	3600.019	35	32	3600.033	35	15	3600.015
jean	0.900	33	33	4	3600.032	33	11	3600.020	33	27	3600.014	33	14	3600.011
jean	0.800	31	30	3	3600.034	31	11	3600.022	30	11	3600.073	30	12	3600.013
jean	0.700	29	29	4	3600.030	29	9	3600.020	29	4	3600.015	27	10	3600.015
jean	0.600	25	24	4	3600.031	23	9	3600.484	24	3	3600.018	20	9	3600.012
jean	0.500	21	21	3	3600.030	18	8	3600.136	21	3	3600.018	13	7	3600.284
jean	0.400	18	18	3	3600.033	8	7	3600.136	14	3	3600.017	9	6	3600.124
jean	0.300	14	12	3	3600.036	5	5	213.099	12	2	3600.016	5	5	323.268
3-FullIns_3	0.999	38	37	37	38.251	38	9	3600.023	37	37	1.785	38	10	3600.013
3-FullIns_3	0.950	38	38	3	3600.161	38	9	3600.024	37	37	2552.954	38	9	3600.019
3-FullIns_3	0.900	38	38	3	3600.194	38	8	3600.023	37	30	3600.018	38	9	3600.021
3-FullIns_3	0.800	37	37	4	3600.208	37	8	3600.038	36	21	3600.025	37	8	3600.013
3-FullIns_3	0.700	36	36	3	3600.039	36	8	3600.030	35	4	3600.020	36	7	3600.013
3-FullIns_3	0.600	17	17	3	3600.034	17	6	3600.022	17	3	3600.020	17	6	3600.014
3-FullIns_3	0.500	13	13	3	3600.033	13	6	3600.023	13	3	3600.021	13	5	3600.013
3-FullIns_3	0.400	10	10	3	3600.035	10	5	3600.022	10	3	3600.016	10	4	3600.013
3-FullIns_3	0.300	7	7	3	3600.033	6	4	3600.161	7	2	3600.017	7	3	3600.014
david	0.999	36	36	36	188.911	36	11	3600.030	36	36	3.142	36	13	3600.021
david	0.950	35	35	4	3600.049	35	10	3600.030	34	34	2999.125	34	12	3600.017
david	0.900	33	33	3	3600.048	33	10	3600.031	33	24	3600.061	33	11	3600.025
david	0.800	31	31	3	3600.061	31	9	3600.032	29	9	3600.022	29	9	3600.017
david	0.700	29	29	3	3600.044	28	8	3600.031	29	3	3600.048	26	8	3600.017
david	0.600	26	26	3	3600.045	24	7	3600.032	24	3	3600.023	18	7	3600.023
david	0.500	23	22	3	3600.048	19	7	3600.032	21	3	3600.027	14	6	3600.017
david	0.400	20	20	3	3600.050	9	6	3600.033	19	2	3600.025	9	5	3600.017
david	0.300	16	13	3	3600.061	6	5	3600.188	14	2	3600.024	6	4	3600.017
myciel6	0.999	52	48	47	3600.331	52	7	3600.041	48	48	2.481	52	8	3600.023
myciel6	0.950	52	52	3	3600.058	52	6	3600.041	48	10	3600.029	51	7	3600.023
myciel6	0.900	52	52	3	3600.063	52	6	3600.040	48	7	3600.029	52	7	3600.020
myciel6	0.800	52	52	3	3600.062	52	5	3600.043	48	3	3600.030	52	6	3600.037
myciel6	0.700	52	52	3	3600.062	52	5	3600.042	52	3	3600.032	52	5	3600.024
myciel6	0.600	30	30	3	3600.065	30	4	3600.042	29	2	3600.032	23	4	3600.030
myciel6	0.500	26	26	3	3600.063	26	4	3600.041	22	2	3600.030	18	4	3600.023
myciel6	0.400	22	22	3	3600.060	22	3	3600.040	22	2	3600.031	12	3	3600.023
myciel6	0.300	18	16	2	3600.065	6	3	3600.042	15	2	3600.031	6	2	3600.022
												C	ontinued on	next page

Table $4 - $ continued from previous page														
		MSH		STD			STDv			REP			REPv	
Input graph	γ	best	best	bbound	time (s)									
4-FullIns_3	0.999	56	55	47	3600.107	56	12	3600.073	55	55	7.057	56	12	3600.039
4-FullIns_3	0.950	56	56	3	3600.403	56	11	3600.073	55	42	3600.054	56	12	3600.038
4-FullIns_3	0.900	56	56	3	3600.110	56	10	3600.075	54	33	3600.056	56	11	3600.038
4-FullIns_3	0.800	54	54	3	3600.111	54	9	3600.076	53	6	3600.056	54	10	3600.039
4-FullIns_3	0.700	53	53	3	3600.108	53	8	3600.074	52	3	3600.140	53	9	3600.038
4-FullIns_3	0.600	26	26	3	3600.109	26	8	3600.092	26	3	3600.059	26	8	3600.037
4-FullIns_3	0.500	21	21	3	3600.102	21	6	3600.080	21	2	3600.051	20	6	3600.050
4-FullIns_3	0.400	16	16	3	3600.104	15	5	3600.079	16	2	3600.060	15	5	3600.052
4-FullIns_3	0.300	10	10	3	3600.104	10	4	3600.072	10	2	3600.057	10	4	3600.041
ieeebus	0.999	58	57	52	3600.243	58	36	3600.079	57	57	4.933	57	57	2117.839
ieeebus	0.950	58	58	3	3600.116	58	36	3600.079	57	57	151.272	57	57	2112.832
ieeebus	0.900	58	57	3	3600.276	58	34	3600.184	57	57	498.775	57	57	1776.998
ieeebus	0.800	58	57	3	3600.246	58	27	3600.080	57	52	3600.075	57	57	2509.315
ieeebus	0.700	58	57	3	3600.114	58	27	3600.080	57	45	3600.069	57	57	3369.871
ieeebus	0.600	40	40	3	3600.114	40	23	3600.078	40	5	3600.061	40	24	3600.044
ieeebus	0.500	34	34	3	3600.116	33	18	3600.084	34	4	3600.058	31	20	3600.045
ieeebus	0.400	28	28	3	3600.118	26	15	3600.085	28	3	3600.109	25	16	3600.049
ieeebus	0.300	21	21	3	3600.117	20	12	3600.081	20	2	3600.146	18	12	3600.044
sfi	0.999	65	65	64	3600.114	65	29	3600.079	65	65	5.061	65	65	764.751
sfi	0.950	65	65	3	3600.121	65	28	3600.079	65	61	3600.056	65	65	1202.593
sfi	0.900	65	65	3	3600.334	65	27	3600.078	65	53	3600.217	65	65	2989.849
sfi	0.800	61	61	3	3600.120	61	24	3600.078	61	43	3600.194	61	36	3600.044
sfi	0.700	59	59	3	3600.135	59	22	3600.079	58	29	3600.059	57	29	3600.046
sfi	0.600	50	50	3	3600.113	48	19	3600.101	49	5	3600.059	45	25	3600.044
sfi	0.500	46	46	3	3600.119	41	17	3600.081	40	3	3600.146	32	21	3600.044
sfi	0.400	41	41	3	3600.119	35	15	3600.084	39	3	3600.062	24	16	3600.049
sfi	0.300	29	29	3	3600.116	16	13	3600.507	23	2	3600.060	16	12	3600.044
anna	0.999	80	80	80	264.461	80	18	3600.167	80	80	9.807	80	20	3600.070
anna	0.950	79	79	3	3600.199	79	17	3600.130	79	63	3600.096	79	19	3600.068
anna	0.900	79	79	3	3600.211	79	16	3600.137	77	54	3600.106	79	18	3600.076
anna	0.800	76	76	3	3600.203	76	15	3600.129	74	13	3600.095	76	16	3600.071
anna	0.700	73	73	3	3600.215	73	13	3600.133	70	3	3600.106	72	14	3600.074
anna	0.600	63	63	3	3600.227	61	11	3600.135	63	3	3600.091	62	12	3600.075
anna	0.500	60	60	3	3600.187	57	10	3600.134	60	2	3600.085	51	10	3600.074
anna	0.400	54	54	3	3600.183	48	9	3600.133	50	2	3600.232	40	8	3600.074
anna	0.300	45	45	3	3600 195	39	7	3600 134	38	2	3600 107	22	6	3600 074

APÊNDICE D – Biased random-key genetic algorithms using path-relinking as a progressive crossover strategy

Biased random-key genetic algorithms using path-relinking as a progressive crossover strategy

Celso C. Ribeiro* celso@ic.uff.br Universidade Federal Fluminense Institute of Computing Niterói, RJ 24271-270, Brazil José A. Riveaux* jangel.riveaux@gmail.com Universidade Federal Fluminense Institute of Computing Niterói, RJ 24271-270, Brazil Julliany S. Brandão* dscjbrandao@gmail.com CEFET/RJ Campus Maracanã Rio de Janeiro, RJ 20271-110, Brazil

ABSTRACT

In a biased random-key genetic algorithm, a deterministic decoder algorithm takes a solution represented by a vector of real numbers (random-keys) and builds a feasible solution for the problem at hand. Selection is said to be biased not only because one parent is always a high-quality solution, but also because it has a higher probability of passing its characteristics to its offspring. Path-relinking is a search intensification strategy to explore trajectories connecting high-quality solutions. In this work, we show how path-relinking can be applied in the space of the random-keys and successfully explored as a progressive crossover strategy in biased random-key genetic algorithms. The efficiency of the newly proposed improved crossover strategies, combining multiple crossover operators with the progressive crossover strategy by path-relinking, is illustrated by applications on two problems: the single-round divisible load scheduling problem and the multi-round divisible load scheduling problem. The computational results show that the improved crossover strategies, combining multiple crossover operators with the progressive crossover strategy by path-relinking, are able not only to improve the running times of the original BRKGA, but also to find better solutions in the same running times.

CCS CONCEPTS

• Theory of computation \rightarrow Theory of randomized search heuristics; Algorithm design techniques; • Applied computing \rightarrow Decision analysis; • General and reference \rightarrow Experimentation.

KEYWORDS

Genetic algorithms, Biased random-key genetic algorithms, Pathrelinking, Progressive crossover, Multiple crossover, Divisible load scheduling

ISMSI 2021, April 10-11, 2021, Victoria, Seychelles

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-8967-9/21/04...\$15.00

ACM Reference Format:

Celso C. Ribeiro, José A. Riveaux, and Julliany S. Brandão. 2021. Biased random-key genetic algorithms using path-relinking as a progressive crossover strategy . In 2021 5th International Conference on Intelligent Systems, Meta-heuristics & Swarm Intelligence (ISMSI 2021), April 10–11, 2021, Victoria, Seychelles. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3461598. 3461603

1 INTRODUCTION

Random-key genetic algorithms (RKGA) were introduced by Bean [3]. Solutions are associated with vectors of real numbers (or keys). A deterministic algorithm, which in this context is also called a decoder, takes a vector of real keys as input to produce a feasible solution of the optimization problem at hand and computes its fitness or objective value. Parents are randomly selected from the entire population for mating and crossover, with repetitions allowed.

In a biased random-key genetic algorithm (BRKGA), once again each solution is represented by a vector of real numbers and a decoder is used to produce feasible solutions.

Selection is said to be biased because one of the two parents selected for mating in each crossover operation is always an elite solution and has a higher probability of passing its genes to the new generation, while the other is a non-elite solution. The reader is referred to Gonçalves and Resende [16] and Resende and Ribeiro [32] for complete reviews about biased random-key genetic algorithms and their applications.

Biased random-key genetic algorithms have been successfully used for solving many combinatorial optimization problems and, in particular, problems that amount to the search for best permutations or best assignments of elements of the ground set (see e.g. [7, 9, 11, 16–19, 26, 27, 29, 36].

Other components, such as local search applied after crossover (see e.g. [8, 27]) and periodical restarts after stabilization (see e.g. [4, 7, 27, 28, 31, 36]), have been explored in the literature to improve the efficiency and the effectiveness of BRKGAs.

Path-relinking is a search intensification strategy to explore trajectories connecting elite (i.e., high-quality) solutions of combinatorial optimization problems [33]. As a major enhancement to heuristic search methods, its hybridization with other metaheuristics has led to significant improvements in both solution quality and running times of hybrid methods.

However, although path-relinking has been successfully applied in hybridizations with several metaheuristics such as tabu search [12], scatter search [13–15, 34], and GRASP [30, 33], it has been scarcely applied in conjunction with genetic algorithms and never with biased random-key genetic algorithms.

^{*}All authors contributed equally to this research. Work of Jose A. Riveaux was supported by CNPq scholarships. Work of Celso C. Ribeiro was partially supported by CNPq research grants 303958/2015-4 and 425778/2016-9 and by FAPERJ research grant E-26/202.854/2017. This work was also partially sponsored by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Finance Code 001.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

https://doi.org/10.1145/3461598.3461603

In this work, we adapt and extend the idea of path-relinking as a progressive crossover operation to the context of biased randomkey genetic algorithms. To the best of our knowledge, this is the first time path-relinking is applied in conjunction with a BRKGA. In Section 2, we review the mechanics and population dynamics of biased random-key genetic algorithms, together with their main building blocks and a framework for their implementation. Section 3 presents improved crossover strategies for biased random-key genetic algorithms, based on multiple crossover operators and on a progressive crossover by path-relinking. The divisible load scheduling problem that is used to assess the efficiency of the newly proposed crossover strategies is summarized in Section 4. Computational experiments illustrating the efficiency of the new strategies are reported in Section 5. Concluding remarks are drawn in the last section.

2 BRKGA: MECHANICS AND POPULATION DYNAMICS

A BRKGA evolves a population of of chromosomes that are formed by vectors of real numbers. Each chromosome of this population is a vector of n components, where n represents the number of decision variables. Each of these components (or keys) is a real number in the interval [0, 1), associated with one of the components of the solution. The chromosome is decoded by an algorithm that takes the vector of real keys as input and outputs a feasible solution for the problem at hand. The decoder is problem specific.

One of the main characteristics of a BRKGA is that each new solution is generated by the combination of one solution selected at random from the subset of elite solutions of the current population with another that is always a non-elite solution. This selection is biased not only because one parent is always an elite solution, but also because it has a higher probability of passing its characteristics to its offspring.

Another central issue of a BRKGA is that all offspring are always feasible solutions, since they are constructed by a decoder that builds feasible solutions from any sequence of random keys.

The parametric uniform crossover of Spears and de Jong [37] is often applied to combine two mates: each offspring inherits with higher probability each of its keys from the best fit of the two parents. Instead of the standard mutation operator, the concept of mutants is used: randomly generated mutant solutions are introduced in the population in each generation. Mutants play the same role of the mutation operator in traditional genetic algorithms, diversifying the search and helping the procedure to escape from locally optimal solutions, see [5, 6, 16, 26, 32]. The initial population is randomly generated.

The current population at any generation is partitioned into two subsets at each generation: *TOP* and *REST*. Subset *TOP* contains the elite solutions, being formed by the best solutions in the population of the current generation. Subset *REST* is decomposed in two subsets: *MID* and *BOT*, with subset *BOT* being formed by the worst elements in the population of the current generation. The size of the population is |TOP| + |REST|.

The population evolves from one generation to the next as illustrated in Figure 1. First, the solutions in *TOP* are simply copied to the population of the next generation. The elements in *BOT* are



Figure 1: Population evolution between consecutive generations of a BRKGA.

replaced by new randomly created mutants that are placed in the new set *BOT*. The remaining |MID| = |REST| - |BOT| solutions of the new population are obtained by crossover, with one parent randomly chosen from *TOP* and the other from *REST*. This is the main difference between a BRKGA and the random-key genetic algorithm of Bean [3], where both parents are randomly selected from the entire population: since a solution can be chosen for mating more than once in any given generation, elite solutions have a higher probability of passing their random keys to the next generation.

Figure 2 illustrates the library brkgaAPI implemented in C++ by Toso and Resende [38], which is a framework for the development of biased random-key genetic algorithms. Its instantiation to some specific optimization problem requires only the development of a class implementing the decoder for the problem at hand, since this is the only problem-dependent part of the tool.

The brkgaAPI framework requires the following parameters [20]: (a) the population size p = |TOP| + |REST|; (b) the fraction *pe* of the population corresponding to the elite subset *TOP*; (c) the fraction *pm* of the population corresponding to the mutant subset *BOT*; (d) the probability *rhoe* that the offspring inherits the keys from the best fit parent; and (e) the number *k* of generations without improvement in the best solution until a restart is performed.

3 BRKGA: IMPROVED CROSSOVER STRATEGIES

In this work, we propose two new strategies to implement extended, improved crossover strategies in the context of biased random-key genetic algorithms.

3.1 Multiple crossover operators

Multiple crossover strategies introduce the possibility that two or more crossover operators are used. For easy of explanation, suppose two crossover operators are used. A parameter $pl \in [0, 1]$ determines the probability the first crossover operator, while 1-pl is the probability the second crossover operator is applied to the selected parents. In consequence, set *MID* in the next generation will be formed by approximately $pl \cdot |MID|$ offspring generated by the first operator and by approximately $(1-pl) \cdot |MID|$ offspring generated by the second. Parent selection rules may be the different or the same for each crossover operator, depending on the application at hand. The rest of the algorithm remains unaltered.

Celso C. Ribeiro, José A. Riveaux, and Julliany S. Brandão

Biased random-key genetic algorithms using path-relinking as a progressive crossover strategy

ISMSI 2021, April 10-11, 2021, Victoria, Seychelles



Figure 2: BRKGA framework.

3.2 Progressive crossover by path-relinking

Ribeiro and Vianna [35] proposed the use of path-relinking as a progressive crossover operation in conventional genetic algorithms: instead of producing only one offspring, it investigates many solutions that share the same characteristics of the selected parents. The solution obtained by applying path-relinking to two parent solutions corresponds to the best offspring that could be obtained by applying the standard crossover to the same parents.

Given two parent solutions (in the space of random-keys), namely the initial solution (S^i) and the guiding solution (S^g), the progressive crossover strategy creates a path $S^0 = S^i, S^1, S^2, \ldots, S^{kmax}$ (in the space of the random-keys), each of them associated with a feasible solution of the problem at hand, such that the randomkeys corresponding to S^{k+1} are progressively similar (i.e., closer) to those corresponding to S^g than those corresponding to S^k , for $k = 0, 1, \ldots, kmax - 1$.

Let the non-elite parent correspond to the initial solution $S^i = \{s_1^i, \ldots, s_n^i\}$ and the elite parent to the guiding solution $S^g = \{s_1^g, \ldots, s_n^g\}$, with $s_1^i, \ldots, s_n^i, s_1^g, \ldots, s_n^g \in [0, 1)$. In addition, let $\sigma \in (0, 1)$ be a step size parameter.

The strategy of progressive crossover by path-relinking creates a path leading from S^i and going towards S^g . The *k*-th solution in this path is $S^k = (s_1^k, \ldots, s_j^k)$, with each s_j^k , for $j = 1, \ldots, n$, computed as

$$s_j^k = s_j^i + k \cdot \sigma \cdot (s_j^g - s_j^i), \tag{1}$$

with $k = 1, \ldots, kmax$ and such that $k \cdot \sigma \leq 1$.

When the path direction is reversed, by exchanging the roles of the two parent solutions, a path from S^{g} to S^{i} is created.

Let *kmax* be the number of solutions in the path between the two parent solutions. Then *kmax* = $\lfloor 1/\sigma \rfloor$ if $1/\sigma$ is not integer, otherwise *kmax* = $1/\sigma - 1$.

Example: Suppose that $S^i = (0.1, 0.5, 0.2, 0.4, 0.2, 0.8)$ is the elite chromosome and $S^g = (0.3, 0.2, 0.8, 0.6, 0.1, 0.4)$ is the non-elite chromosome. Then, $S^g - S^i = (0.2, -0.3, 0.6, 0.2, -0.1, -0.4)$. If the step size is $\sigma = 0.4$, then kmax = 2 and the chromosomes that define the path from S^i to S^g are $(0.1, 0.5, 0.2, 0.4, 0.2, 0.8) + 1 \cdot 0.4 \cdot (0.2, -0.3, 0.6, 0.2, -0.1, -0.4)$ and $(0.1, 0.5, 0.2, 0.4, 0.2, 0.8) + 1 \cdot 0.4 \cdot (0.2, -0.3, 0.6, 0.2, -0.1, -0.4)$, i.e., (0.18, 0.38, 0.44, 0.48, 0.16, 0.64) and (0.26, 0.26, 0.68, 0.56, 0.12, 0.48). However, if we change the roles of the elite and non-elite solutions, the chromosomes that define the path from S^i to S^g are $(0.3, 0.2, 0.8, 0.6, 0.1, 0.4) + 1 \cdot 0.4 \cdot (-0.2, 0.3, -0.6, -0.2, 0.1, 0.4) = (0.22, 0.32, 0.56, 0.52, 0.14, 0.56)$ and $(0.3, 0.2, 0.8, 0.6, 0.1, 0.4) + 2 \cdot 0.4 \cdot (-0.2, 0.3, -0.6, -0.2, 0.1, 0.4) = (0.14, 0.44, 0.32, 0.44, 0.44, 0.18, 0.72).$

The new extension of biased random-key genetic algorithm implementing the improved crossover strategies described in this section has the same structure of the brkgaAPI in Figure 2. The user only needs to develop the decoder for the problem at hand and to configure the input parameters. It takes two additional types of parameters: (f) the number of crossover operators that are used and the probability each of them is applied; and (g) the path-relinking parameter *stepsize*. This new api will be made available for free use in a forthcoming publication.

4 DIVISIBLE LOAD SCHEDULING

A divisible load is an amount $W \ge 0$ of computational work that can be arbitrarily divided and distributed among different processors to be processed in parallel. The processors are arranged in a star topology and the load is stored in a central master processor. The master splits the load into chunks of arbitrary sizes and transmits each of them to other worker processors (or, simply, processors).

The master can only send load to one processor at a time. It is assumed that it does not process the load itself. Any processor can only start processing after it has completely received its respective chunk of the load. The processors are heterogeneous in terms of

processing power, communication speed, and setup time for start communicating with the master. Not all available processors should necessarily be used for processing the load. Consequently, despite how the load is split, the choice of (i) which processors are used and (ii) the order in which the chunks are transmitted influences the total processing time of the load.

The Divisible Loading Scheduling (DLS) problem was introduced in [10], motivated by an application in intelligent sensor networks. Applications of DLS arise from a number of scientific problems, such as parallel database searching [22], parallel image processing [23], parallel video encoding [24, 39], processing of large distributed files [40], and task scheduling in cloud computing [25], among others.

Brandão et al. [5] dealt with the single-round variant SR-DLS of the problem, in which each active processor receives and processes a single chunk of the load. Let $W \in \mathbb{R}$ be the amount of load to be processed, 0 (zero) be the index of the master processor, and P = $\{1, ..., n\}$ be the set of worker processors indices. The scheduling problem consists of (a) selecting a subset $A \subseteq P$ of active processors, (b) defining the order in which the chunks will be transmitted to each active processor and (c) deciding the amount of load α_i that will be transmitted to each processor $i \in A$, with $\sum_{i=1}^{n} \alpha_i = W$, so as to minimize the makespan, i.e., the total elapsed time since the master began to send data to the first processor, until the last processor stops its computations. An example of an optimal singleround schedule is shown in Figure 3. The biased random-key genetic algorithm proposed in [5] outperformed the existing heuristics for DLS-SR at the time of publication.



Figure 3: Example of an optimal single-round schedule.

In multi-round systems, data transmission from the master is divided into rounds and every active worker receives a load chunk at each round. The activation order is the same for all rounds. Multi-round systems may reduce the makespan by decreasing the idle times, at the additional cost of incurring in repeated setup times at each round. In this case, the multi-round variant MR-DLS of the divisible load scheduling problem consists in (a) determining the activation order of the processors, (b) deciding the number n of active workers, (c) choosing the number m of rounds, and (d) determining the amount of load θ_i^k that will be transmitted to each processor $i \in P$ at each round $k \in \{1, ..., m\}$, with $\sum_{i \in P} \sum_{k \in \{1,...,m\}} \theta_i^k = W$, so as to minimize the makespan. Figure 4 illustrates an optimal three-round schedule. The BRKGA heuristic proposed by Brandão et al. [6] for MR-DLS improved upon the best existing algorithm in the literature in terms of solution quality, since it performs a global search in the space of processor permutations in order to find the best transmission order of the processors

Celso C. Ribeiro, José A. Riveaux, and Julliany S. Brandão

Table 1: BRKGA* configurations.

Configuration	pl	stepsize
1	0.10	0.10
2	0.10	0.20
3	0.10	0.30
4	0.10	0.50
5	0.25	0.10
6	0.25	0.20
7	0.25	0.30
8	0.25	0.50

and the best values for the number of active processors and for the number of rounds.



Figure 4: Example of an optimal three-round schedule.

5 COMPUTATIONAL EXPERIMENTS

The improved crossover strategies proposed in Section 3 have been applied to both the single-round (SR-DLS) and the multi-round (MR-DLS) variants of the divisible load scheduling problem described in Section 4. In each case, we used the same decoders originally described in [5] and [6], respectively, to create extended variants of the original BRKGAs incorporating the improved crossover strategies.

All algorithms were implemented using version 18.00.21005.1 of the Microsoft®C/C++ Optimizing Compiler for x64. The computational experiments have been performed on an Intel Core i5-5200 processor with 2.20 GHz and 8 GB of RAM running under Windows 10.

Table 1 show all eight configurations of algorithm BRKGA^{*} implementing the improved crossover strategies, combining different values of the parameter *stepsize* with two types of crossover: the progressive crossover by path-relinking proposed in Section 3.2 with probability pl and that in the original reference with probability 1 - pl, for both problems SR-DLS and MR-DLS.

5.1 Experiments with SR-DLS

We selected 60 amongst the hardest and largest test instances of SR-DLS with 160 workers, also used in the experiments reported in [7]. Each algorithm (the original BRKGA and the new BRKGA* with improved crossover strategies) was run ten times for each instance for ten minutes. For all test instances, all configurations of BRKGA*, except configuration 7, and the original BRKGA obtained the same best solution.

ISMSI 2021, April 10-11, 2021, Victoria, Seychelles

Table 2 summarizes the results obtained with each parameter configuration of BRKGA*, comparing them with the original BRKGA. For each configuration of BRKGA*, column "faster" shows for how many instances it reached the best solution in average time lower than the original BRKGA, while column "slower" indicates the opposite and "ties" indicates the number of instances where both algorithms obtained the best solution in the same average time. Column "winning difference" is the sum of the differences of the average times in seconds over all instances for which BRKGA* was faster, while "losing difference" denotes the sum of the differences of the average times in seconds over all instances where it was slower. The "average winning time" is the ratio between the winning difference and the number of instances for which BRKGA* was faster, while the "average losing time" indicates the opposite. The last two columns indicate the number of instances for which the average time taken by BRKGA* to reach the best solution was smaller than that of the original BRKGA by a difference greater than one second ("faster") and the opposite ("slower").

Table 2 shows that configurations 2, 3, 4, and 8 beat the original BRKGA in terms of the number of wins, i.e., smaller average times over ten runs. We observe that, for all configurations of BRKGA*, the average winning difference in seconds is at least four times the average losing difference in seconds. In other words, the gains in the instances where BRKGA* was faster than BRKGA were much more significant than the losses in the instances for which BRKGA performed better.

Furthermore, the last two columns show that if one considers only the instances where the winning or losing differences are greater than one second, then all configurations of BRKGA* beat the original BRKGA in terms of the number of wins, i.e., smaller average times over ten runs.

Run time distributions (or time-to-target plots) display on the vertical axis the probability that some algorithm will find a solution at least as good as a given target within some given processing time, which appears in the abscissa axis. They have also been claimed by Hoos and Stützle [21] as a very effective tool to characterize the behavior of running times of stochastic local search algorithms. In the experiments reported in the sequel, the algorithms are made to stop whenever a solution with cost smaller than or equal to a given target is found. Each heuristic was run 200 times, with different initial seeds for the pseudo-random number generator, and the empirical probability distribution of the time taken by each heuristic to find a solution at least as good as the target is plot. We followed the methodology proposed by Aiex et al. [1, 2] to build the empirical distributions. A probability $p_i = (i - \frac{1}{2})/200$ is associated with the *i*-th smallest running time t_i and the points (t_i, p_i) are plotted, for i = 1, ..., 200. The more to the left is a plot, the better is the corresponding algorithm.

In the following, we compare the run time distributions (or timeto-target plots or, more simply, tttplots) of configurations 2, 3, 4, and 8 of algorithm BRKGA* with those of the original BRKGA algorithm on the instances with the largest average winning time differences and the largest average losing time differences in Table 2, which are listed in Table 3. The table also displays the target values, which were set as the cost of the best known solution for each instance. Instance I-2 presented the largest average losing time differences



Figure 5: Time-to-target plots for instances I-3 and I-4, showing that the BRKGA* configurations implementing the improved crossover strategies performed significantly better than the original BRKGA algorithm, finding the targets much faster in all cases.

for all four configurations of BRKGA*, followed by instance I-1 for configurations 2 and 8 and by I-5 for configuration 3 and 4. Instances I-3 and I-4 showed the largest average winning time differences for the four BRKGA* configurations.

Figure 5 shows the time-to-target plots for instances I-3 and I-4, which are those with the largest average winning time differences. In the case of instance I-3, all four configurations 2, 3, 4, and 8 of BRKGA* reached the target before 150 seconds with probabilities 98.17%, 99.10%, 97.51%, and 98.68%, respectively, while BRKGA did it with a probability of only 78.69%. Similarly, for instance I-4, all four configurations 2, 3, 4, and 8 of BRKGA* reached the target before 40 seconds, with probabilities 99.25% for the three first and 98.76% for the last, while BRKGA did it with with a probability that is not greater than 67.01%.

Figure 6 displays the tttplots for instances I-1, I-2, and I-5. Even though these were the instances with the largest average losing time differences, we notice that only for instance I-2 the run time distribution of BRKGA was only marginally better than that of the best configuration of BRKGA*: BRKGA reached the target before 100 seconds with probability 97.34%, while configuration 4 of BRKGA* (that beat the other three configurations) did it with probability 90.81%. For instance I-1, BRKGA and the four configurations of BRKGA* performed remarkably similarly. Finally, for instance I-5, all configurations of BRKGA* performed better than BRKGA: while BRKGA reached the target before 10 seconds with probability 85.54%, configurations 2, 3, 4, and 8 did it with larger probabilities: 99.25%, 99.25%, 97.77%, and 97.94%, respectively.

5.2 Experiments with MR-DLS

For the computational experiments with the divisible load scheduling problem MR-DLS, we considered the same six instances used in the numerical results reported in [7]. They all have 50 workers

Table 2: Comparative results for BRKGA* on problem SR-DLS over 60 instances with 160 workers (ten runs each).

	Faster	Slower	Ties	Winning	Losing	Avg. winning	Avg. losing	Faster	Slower
Config.	(avg.)	(avg.)	(avg.)	difference (s)	difference (s)	difference (s)	difference (s)	> 1s (avg.)	> 1s (avg.)
1	16	43	1	353.478125	172.495313	22.092383	4.011519	10	3
2	31	28	1	408.937500	61.554688	13.191532	2.198382	11	3
3	31	28	1	418.965625	37.976562	13.515020	1.356306	11	1
4	38	18	4	387.843750	42.582812	10.206414	2.365712	12	2
5	11	49	0	298.878125	220.092188	27.170739	4.491677	8	5
6	20	40	0	386.129687	78.167187	19.306484	1.954180	12	2
7	21	37	1	268.575000	40.332812	12.789286	1.090076	10	3
8	32	26	2	448.943750	50.621875	14.029492	1.946995	12	3

Table 3: Instances with the largest average time differences selected for the experiments with run time distributions.

Nickname	Original instance name in [5]	Target value
I-1	Grid1_160p.HET.1788LO_wLO_gLO_G.id_01.W1600.grid	7010.502813
I-2	Grid1_160p.HET.1788LO_wLO_gLO_G.id_01.W200.grid	941.456261
I-3	Grid1_160p.HET.1788LO_wLO_gLO_G.id_01.W3200.grid	13853.374469
I-4	Grid1_160p.HET.1814LO_wLO_gLO_G.id_03.W3200.grid	16087.057779
I-5	Grid1_160p.HET.1814LO_wLO_gLO_G.id_03.W400.grid	2312.624185

and differ by their loads, which are W = 50, 250, 450, 650, 850, and 1050.

Each algorithm (the original BRKGA genetic algorithm and the eight parameter configurations of the new BRKGA* with improved crossover strategies) was run 200 times with different seeds for each instance, until a target solution value was reached. The selected target values were: 1108.19 (W = 50), 3952.01 (W = 250), 6591.64 (W = 450), 9223.45 (W = 650), 11854.06 (W = 850), and 14484.13 (W = 1050).

Table 4 shows, from top (fastest) to bottom (slowest), the sum of the average times-to-target values over the six test instances for each algorithm version. It shows that configuration 3 of the new BRKGA* algorithm performed clearly better, with the smallest sum of the average times-to-target values. In particular, the total average time for configuration 3 of BRKGA* amounts to only 31% of the original BRKGA.

Table 4: Sum of the average times-to-target values over the six test instances for each algorithm version.

Avg. time	Algorithm
(seconds)	version
30.5109	BRKGA*-3
33.8849	BRKGA*-6
36.4380	BRKGA*-8
37.9572	BRKGA*-2
46.7343	BRKGA*-1
54.8419	BRKGA*-4
66.9288	BRKGA*-5
79.9640	BRKGA*-7
98.1084	BRKGA

In the next experiment, Figures 7 and 8 display the time-totarget plots for the best configuration 3 of BRKGA* and the original

Table 5: Average solution values over ten runs for instances with W = 50, 250, and 450.

Algorithm			
Aigorithin			
version	W = 50	W = 250	W = 450
BRKGA	1108.1368	3951.8267	6591.6438
BRKGA*-1	1108.0926	3951.0268	6591.6438
BRKGA*-2	1108.0905	3950.9727	6591.6421
BRKGA*-3	1108.0706	3950.9462	6591.6292
BRKGA*-4	1108.0777	3950.8987	6591.6410
BRKGA*-5	1108.1534	3954.3421	6591.6620
BRKGA*-6	1108.0738	3951.6608	6591.6302
BRKGA*-7	1108.1158	3951.4859	6591.6421
BRKGA*-8	1108.1296	3951.0110	6591.6347

BRKGA for the six test instances. These plots show unarguably that the proposed BRKGA* (in particular, when configuration 3 is used for the case of MR-DLS) with the new improved crossover strategies significantly improves upon the original BRKGA algorithm.

For the final experiment, BRKGA and the eight configurations of BRKGA^{*} were run ten times each for each instance with the time limits displayed in Table 4. The results are shown in Tables 5 and 6. Tables 5 shows, for each algorithm version, the average solution value over the ten runs for the instances with W = 50, 250, and 450, while Table 6 shows the same results for the instances with W = 650, 850 and 1050. It can be observed that all versions of BRKGA^{*} (except configuration 5 for some instances) found solutions whose average values are better than or equal to those obtained by the original BRKGA algorithm.

Together with the run time distributions illustrated in Figures 7 and 8, the results in Tables 5 and 6 show that the improved crossover strategies, combining multiple crossover operators with the progressive crossover strategy by path-relinking, are able not only to



Figure 6: Time-to-target plots for instances I-2, I-1, and I-5, showing that the BRKGA* configurations implementing the improved crossover strategies also performed well for the harder instances.

Figure 7: Comparative run time distributions for the three instances of MS-DLS with the smallest loads.



Figure 8: Comparative run time distributions for the three instances of MS-DLS with the largest loads.

Celso C. Ribeiro, José A. Riveaux, and Julliany S. Brandão

Table 6: Average solution values over ten runs for instances with W = 650, 850, and 1050.

Algorithm			
version	W = 650	W = 850	W = 1050
BRKGA	9223.4526	11854.0606	14484.1329
BRKGA*-1	9223.4455	11854.0537	14484.1273
BRKGA*-2	9223.4460	11854.0531	14484.1273
BRKGA*-3	9223.4458	11854.0531	14484.1273
BRKGA*-4	9223.4498	11854.0533	14484.1296
BRKGA*-5	9225.1626	11856.7469	14484.1273
BRKGA*-6	9223.4453	11854.0531	14484.1273
BRKGA*-7	9223.4470	11854.0584	14484.1296
BRKGA*-8	9223.4460	11854.0537	14484.1278

improve the running times of the original BRKGA, but also to find better solutions in the same running times.

6 CONCLUDING REMARKS

In a biased random-key genetic algorithm, each solution is represented by a vector of real numbers (or keys). A deterministic algorithm, called a decoder, takes a solution represented by a vector of random keys and builds a feasible solution for the problem at hand. Selection is said to be biased not only because one parent is always a high-quality solution, but also because it has a higher probability of passing its characteristics to its offspring.

Path-relinking is a search intensification strategy to explore trajectories connecting elite (i.e., high-quality) solutions of combinatorial optimization problems.

Although path-relinking has been already hybridized with other metaheuristics, in this work we showed for the first time in the literature how path-relinking can be applied in the space of the random keys and successfully explored as a progressive crossover strategy in biased random-key genetic algorithms.

The efficiency of the newly proposed improved crossover strategies, combining multiple crossover operators with the progressive crossover strategy by path-relinking, is illustrated by applications on two problems: the single-round divisible load scheduling problem and the multi-round divisible load scheduling problem.

Computational experiments and numerical results for both problems on benchmark test instances illustrate the efficiency and the effectiveness of the newly proposed approach. Preliminary results obtained for the maximum clique problem [28] reinforce this conclusion. We are currently working on the application of this approach to other optimization problems using benchmark test instances.

REFERENCES

- R. M. Aiex, M. G. C. Resende, and C. C. Ribeiro. 2002. Probability Distribution of Solution Time in GRASP: An Experimental Investigation. *Journal of Heuristics* 8 (2002), 343–373.
- [2] R. M. Aiex, M. G. C. Resende, and C. C. Ribeiro. 2007. TTTPLOTS: A Perl program to create time-to-target plots. *Optimization Letters* 1 (2007), 355–366.
- [3] J. C. Bean. 1994. Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 2 (1994), 154–160.
 [4] I. S. Brandão, T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro, 2015. A
- [4] J. S. Brandão, T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro. 2015. A biased random-key genetic algorithm for single-round divisible load scheduling. *International Transactions in Operational Research* 22 (2015), 823–839.
- [5] J. S. Brandão, T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro. 2015. A biased random-key genetic algorithm for single-round divisible load scheduling.

International Transactions Operational Research 22 (2015), 823-839.

- [6] J. S. Brandão, T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro. 2017. A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems. *International Transactions Operational Research* 27 (2017), 1061–1077.
- [7] J. S. Brandão, T. F. Noronha, M. G. C. Resende, and C. C Ribeiro. 2017. A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems. International Transactions in Operational Research 24 (2017), 1061–1077.
- [8] L. S. Buriol, M. G. C. Resende, C. C. Ribeiro, and M. Thorup. 2005. A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. *Networks* 46 (2005), 36–56.
- [9] F. Carrabs. 2021. A Biased Random-Key Genetic Algorithm for the Set Orienteering Problem. *European Journal of Operational Research* (2021). https: //doi.org/10.1016/j.ejor.2020.11.043 https://doi.org/10.1016/j.ejor.2020.11.043.
- [10] Y. C. Cheng and T. G Robertazzi. 1988. Distributed computation with communication delay. *IEEE Trans. Aerospace Electron. Systems* 24 (1988), 700–712.
- [11] A. Duarte, R. Martí, M. G. C. Resende, and R. M. A. Silva. 2014. Improved heuristics for the regenerator location problem. *International Transactions in Operational Research* 21 (2014), 541–558.
- [12] F. Glover. 1997. Tabu Search and Adaptive Memory Programming Advances, Applications and Challenges. In Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies, Richard S. Barr, Richard V. Helgason, and Jeffrey L. Kennington (Eds.). Springer, Boston, 1–75.
- [13] F. Glover, M. Laguna, and R. Martí. 2000. Fundamentals of scatter search and path relinking. *Control and Cybernetics* 39 (2000), 653–684.
- [14] F. Glover, Manuel Laguna, and Rafael Marti. 2003. Scatter Search and Path Relinking: Advances and Applications. In *Handbook of Metaheuristics*, F. Glover and Gary A. Kochenberger (Eds.). Springer, Boston, 1–35.
- [15] F. Glover, M. Laguna, and R. Martí. 2004. Scatter Search and Path Relinking: Foundations and Advanced Designs. In *New optimization techniques in engineering*, G.C. Onwubolu and B.V. Babu (Eds.). Studies in Fuzzyness and Soft Computing, Vol. 141. Springer, Berlin, 87–100.
- [16] J. F. Goncalves and M. G. C. Resende. 2011. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics* 17 (2011), 487–525.
- [17] J. F. Gonçalves and M. G. C. Resende. 2013. A biased random key genetic algorithm for 2D and 3D bin packing problems. *International Journal of Production Economics* 145 (2013), 500–510.
- [18] J. F. Gonçalves and M. G. C. Resende. 2014. An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. *International Transactions in Operational Research* 21 (2014), 215–246.
- [19] J. F. Gonçalves, M. G. C. Resende, and M. D. Costa. 2016. A biased random-key genetic algorithm for the minimization of open stacks problem. *International Transactions in Operational Research* 23 (2016), 25–46.
- [20] J. F. Goncalves, M. G. C. Resende, and R. F. Toso. 2013. Biased and unbiased random key genetic algorithms: An experimental analysis. In Abstracts of the 10th Metaheuristics International Conference. Singapore.
- [21] H. H. Hoos and T. Stützle. 1998. Evaluation of Las Vegas algorithms Pitfalls and remedies. In Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, G. Cooper and S. Moral (Eds.). Madison, 238–245.
- [22] M. Drozdowskiand M. Markiewicz J. Błażewicz. 1999. Divisible task schedulingconcept and verification. *Parallel Comput.* 25 (1999), 87–98.
- [23] C.-K. Lee and M. Hamdi. 1995. Parallel image processing applications on a network of workstations. *Parallel Comput.* 21 (1995), 137–160.
- [24] Ping Li, Bharadwaj Veeravalli, and Ashraf A Kassim. 2005. Design and implementation of parallel video encoding strategies using divisible load analysis. *IEEE Transactions on Circuits and Systems for Video Technology* 15 (2005), 1098–1112.
 [25] W. Lin, C. Liang, J. Z. Wang, and R. Buyya. 2014. Bandwidth-aware divisible
- [25] W. Lin, C. Liang, J. Z. Wang, and R. Buyya. 2014. Bandwidth-aware divisible task scheduling for cloud computing. *Software: Practice and Experience* 44 (2014), 163–174.
- [26] T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro. 2011. A biased random-key genetic algorithm for routing and wavelength assignment. *Journal of Global Optimization* 50 (2011), 503–518.
- [27] B. Q. Pinto, C. C. Ribeiro, J.-A. Riveaux, and I. Rosseti. 2021. A BRKGA-based matheuristic for the maximum quasi-clique problem with an exact local search strategy. *RAIRO Operations Research* (2021).
- [28] B. Q. Pinto, C. C. Ribeiro, I. Rosseti, and A. Plastino. 2018. A biased random-key genetic algorithm for solving the maximum quasi-clique problem. *European Journal of Operational Research* 271 (2018), 849–865.
- [29] M. G. C. Resende. 2012. Biased random-key genetic algorithms with applications in telecommunications. TOP 20 (2012), 130–153.
- M. G. C. Resende and C. C. Ribeiro. 2005. GRASP with path-relinking: Recent advances and applications. In *Metaheuristics: Progress as real problem solvers*, T. Ibaraki, K. Nonobe, and M. Yagiura (Eds.). Springer, New York, 29–63.
- [31] M. G. C. Resende and C. C. Ribeiro. 2011. Restart strategies for GRASP with path-relinking heuristics. *Optimization Letters* 5 (2011), 467–478.
- [32] M. G. C. Resende and C. C. Ribeiro. 2016. Biased-random key genetic algorithms: An advanced tutorial. In Proceedings of the 2016 Genetic and Evolutionary Computation Conference - GECCO'16 Companion Volume. Association for Computing

Machinery, Denver, 483-514.

- [33] M. G. C. Resende and C. C. Ribeiro. 2016. Optimization by GRASP: Greedy Randomized Adaptive Search Procedures. Springer, New York.
- [34] M. G. C. Resende, C. C. Ribeiro, F. Glover, and R. Martí. 2010. Scatter search and path-relinking: Fundamentals, advances, and applications. In *Handbook of metaheuristics* (2nd ed.), M. Gendreau and J.-Y. Potvin (Eds.). Springer, New York, 87–107.
- [35] C. C. Ribeiro and D. S. Vianna. 2009. "A hybrid genetic algorithm for the phylogeny problem using path-relinking as a progressive crossover strategy". *International Transactions in Operational Research* 16 (2009), 641–657.
- [36] L. C. R. Soares and M. A. M. Carvalho. 2020. Biased random-key genetic algorithm for scheduling identical parallel machines with tooling constraints. *European Journal of Operational Research* 285 (2020), 955–964. https://doi.org/10.1016/j. ejor.2020.02.047
- [37] W. Spears and K. de Jong. 1991. On the virtues of parameterized uniform crossover. In Proceedings of the Fourth International Conference on Genetic Algorithms, R. Belew and L. Booker (Eds.). Morgan Kaufman, San Mateo, 230–236.
- [38] R. F. Toso and M. G. C. Resende. 2015. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software* 30 (2015), 81–93.
- [39] A. Turgay and P. Yakup. 2002. Optimal Scheduling Algorithms for Communication Constrained Parallel Processing. *Lecture Notes in Computer Science* 2400 (2002), 197–206.
- [40] R.Y. Wang, A. Krishnamurthy, R.P. Martin, T.E. Anderson, and D.E. Culler. 1998. Modeling communication pipeline latency. ACM Sigmetrics Performance Evaluation Review 26 (1998), 22–32.