UNIVERSIDADE FEDERAL FLUMINENSE

BRUNO QUEIROZ PINTO

ALGORITMOS GENÉTICOS COM CHAVES ALEATÓRIAS TENDENCIOSAS E APLICAÇÕES

NITERÓI

2018

UNIVERSIDADE FEDERAL FLUMINENSE

BRUNO QUEIROZ PINTO

ALGORITMOS GENÉTICOS COM CHAVES ALEATÓRIAS TENDENCIOSAS E APLICAÇÕES

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: Algoritmos e Otimização

Orientador:
CELSO DA CRUZ CARNEIRO RIBEIRO

Co-orientador:
ISABEL CRISTINA MELLO ROSSETI

NITERÓI

2018

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

P659a Pinto, Bruno Queiroz
Algoritmos Genéticos com Chaves Aleatórias Tendenciosas e Aplicações / Bruno Queiroz Pinto ; Celso Carneiro Ribeiro, orientador ; Isabel Cristina Mello Rosseti, coorientadora. Niterói, 2018.
107 f. : il.

Tese (doutorado)-Universidade Federal Fluminense, Niterói, 2018.

DOI: http://dx.doi.org/10.22409/PGC.2018.d.03595264622

1. Metaheurística. 2. Heurística. 3. Produção intelectual. I. Ribeiro, Celso Carneiro, orientador. II. Rosseti, Isabel Cristina Mello, coorientadora. III. Universidade Federal Fluminense. Escola de Engenharia. IV. Título.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

BRUNO QUEIROZ PINTO

Algoritmos Genéticos com Chaves Aleatórias Tendenciosas e Aplicações

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: Algoritmos e Otimização

BANCA EXAMINADORA

Prof. Celso Carneiro Ribeiro - Orientador, UFF

Profa. Isabel Cristina Mello Rosseti - Co-orientador, UFF

Prof. Alexandre Plastino, UFF

Prof. José Viterbo Filho, UFF

Profa. Julliany Sales Brandão, Cefet-RJ

Prof. Thiago Ferreira de Noronha, UFMG

Niterói

2018

"O que somos é a consequência do que pensamos". (Buda)

A Deus, aos meus pais Domingos e Claudetis, a minha irmã Erica e as minhas sobrinhas Clara e Laura. Dedico.

Agradecimentos

Agradeço a Deus e a Nossa Senhora Aparecida.

Aos meus pais Domingos e Claudetis, minha irmã Erica e minhas sobrinhas Laura e Clara, que nunca deixaram de acreditar em mim e em meus sonhos. Sempre me apoiaram e fizeram de tudo para que eu pudesse ter a oportunidade de estudar e chegar até aqui. Vocês são meus amigos e exemplos. Esta conquista é nossa. Amo vocês!

Aos meus amigos e familiares que sempre torceram e oraram por mim.

Aos meus orientadores Celso da Cruz Carneiro Ribeiro e Isabel Cristina Mello Rosseti por extraírem o meu melhor. Tenho muito orgulho de ter trabalhado com cientistas e pessoas admiráveis como vocês.

Ao Wilton de Paula Filho pelas discussões, apoio e atenção.

À Teresa Cancela por sua atenção.

Obrigado a Alexandre Plastino, Thiago Ferreira de Noronha e José Angel Riveaux Merino que contribuíram e apoiaram na realização de deste trabalho.

Resumo

Esta tese relata, na forma de uma coletânea de três artigos, aplicações de heurísticas baseadas em algoritmos genéticos com chaves aleatórias tendenciosas (BRKGA) em dois diferentes problemas de otimização: quasi-clique de cardinalidade máxima (MQCP) e minimização do número total de comprimentos de onda necessários para rotear demandas de caminhos óticos programadas (RWA-SSLD). Para o primeiro problema, as variantes de BRKGA propostas obtiveram melhores resultados quando comparadas aos obtidos pela heurística estado-da-arte da literatura. Já para o RWA-SSLD, os resultados do BRKGA proposto superaram os alcançados por uma heurística construtiva multipartida, e os obtidos pela heurística estado-da-arte da literatura. Todos os dados de entrada para as instâncias utilizadas nos experimentos computacionais relatados na tese e nos artigos associados estão disponíveis na plataforma Mendeley.

Palavras-chave: Algoritmos genéticos com chaves aleatórias tendenciosas; metaheurísticas; heurísticas; quasi-clique de cardinalidade máxima; minimização do número total de comprimentos de onda necessários para rotear demandas programadas.

Abstract

This dissertation reports, in the form of a collection of three articles, applications of biased random key genetic algorithms (BRKGA) to two different optimization problems: the maximum cardinality quasi-clique problem (MQCP) and routing and wavelength assignment under a sliding scheduled traffic model in WDM optical networks (RWA-SSLD). With respect to the first problem, the proposed BRKGA variants outperformed state-of-the-art heuristics in the literature. Regarding RWA-SSLD, the results reached by the biased random-key genetic algorithm outperformed those achieved by a multi-start constructive heuristic and those obtained by a state-of-the-art heuristic in the literature. All the input data for the test instances used in the computational experiments reported in this dissertation and in the associated articles are available in the Mendeley repository.

Keywords: Biased random-key genetic algorithms; metaheuristics; heuristics; maximum quasi-clique problem; routing and wavelength assignment under a sliding scheduled traffic model.

Lista de Figuras

1.1 Fluxo de controle do algoritmo genético. 2

1.2 Combinação uniforme parametrizada. 3

1.3 Ilustração do processo de transição entre duas gerações. 4

1.4 Framework BRKGA. 5

2.1 (a) Grafo G; (b) Grafo induzido em G por $C*$. 10

3.1 Exemplo de topologia de rede. 16

3.2 Modelo de janela de tempo deslizante. 16

3.3 Exemplo de uma janela de tempo deslizante para a demanda d_1 da Tabela 3.1. 17

Lista de Tabelas

2.1 Subconjuntos de vértices de G 10

3.1 Exemplo de problema RWA-SSLD com $m = 4$ demandas. 16

Sumário

1 Introdução **1**

1.1 Aplicações de BRKGA 4

1.2 Estrutura do Trabalho 8

2 Problema quasi-clique de cardinalidade máxima **9**

2.1 Heurísticas e trabalhos relacionados 10

2.2 Contribuições 12

3 Problema de minimização do número total de comprimentos de onda necessários para rotear demandas programadas **14**

3.1 O problema RWA-SSLD 15

3.2 Revisão da literatura 16

3.3 Contribuições 18

4 Conclusões **20**

Referências **22**

Anexo A – Pinto, B. Q., Ribeiro, C. C., Rosseti, I., Plastino, A. "A biased random-key genetic algorithm for the maximum quasi-clique problem". European Journal of Operational Research (2018), p. 849 — 865.

Anexo B – Pinto, B. Q., Ribeiro, C. C., Riveaux, J. A., Rosseti, I. "Improving a biased random-key genetic algorithm for the maximum quasi-clique problem with an exact local search". Submetido para International Transactions in Operational Research.

Anexo C – Pinto, B. Q., Ribeiro, C. C., Rosseti, I., Noronha, T. F. "A biased random-key genetic algorithm for routing and wavelength assignment under a sliding scheduled traffic model". Submetido para Journal of Global Optimization.

Capítulo 1

Introdução

Metaheurísticas são procedimentos genéricos para a solução aproximada de problemas de otimização combinatória. Entre elas, podem ser citadas busca tabu, *simulated annealing*, GRASP, colônias de formigas, algoritmos genéticos e outras. Cada uma delas está baseada em um paradigma distinto e oferece diferentes mecanismos que permitem escapar de ótimos locais, contrariamente aos algoritmos construtivos ou de melhoria.

Algoritmos genéticos (AG), propostos por John Holland [25] em 1975, baseiam-se em princípios da biologia evolutiva, no qual os indivíduos mais aptos possuem maior probabilidade de se reproduzirem. As soluções são representadas por indivíduos (cromossomos), agrupados em uma população que evolui a cada geração por meio do cruzamento entre seus indivíduos, permitindo recombinar os cromossomos e gerar novos, e da mutação, que aleatoriamente modifica genes presentes em alguns cromossomos, de modo a diversificar a nova população e evitar que o processo de evolução fique estagnado. A aptidão de um indivíduo é dada pelo seu *fitness*, cujo valor é resultado da aplicação de uma função de avaliação. O algoritmo é repetido até que um critério de parada seja satisfeito.

Um algoritmo genético pode ser descrito pelo fluxograma apresentado na Figura 1.1. Inicialmente uma população é criada. A aptidão de cada indivíduo é calculada. Com base nesse valor de aptidão, são selecionados indivíduos pais que participarão do processo de geração dos descendentes por meio das operações de cruzamento e de mutação. Ao fim desse processo, a população é atualizada com os novos indivíduos. Esse procedimento é repetido até que um critério de parada seja satisfeito.

Algoritmos genéticos com chaves aleatórias (RKGA) foram primeiramente propostos por Bean [8], para problemas de otimização combinatória, cujas as soluções são representadas por vetores de números reais no intervalo contínuo [0,1), sendo esses números

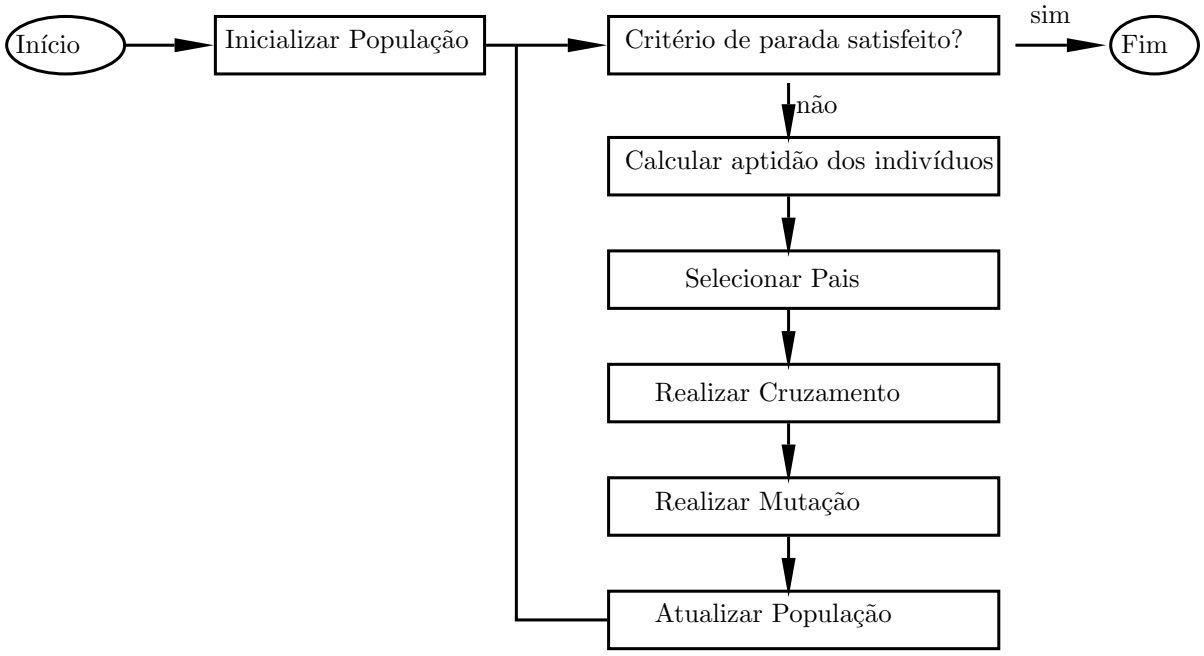


Figura 1.1: Fluxo de controle do algoritmo genético.

chamados de chaves aleatórias, geradas aleatoriamente na população inicial. Um algoritmo determinístico, nomeado como decodificador, recebe como entrada um vetor de chaves aleatórias (cromossomo) e realiza o mapeamento dessas chaves em uma solução do problema de otimização combinatória. Com o mapeamento das chaves em uma solução é possível calcular o seu valor de aptidão. Em um RKGA, a mutação é realizada por meio do conceito de mutantes na população. Os mutantes são vetores de chaves aleatórias, também gerados aleatoriamente no intervalo [0,1). Um pequeno número de mutantes são introduzidos na população em cada geração, prevenindo a convergência prematura do método para ótimos locais [8]. Um RKGA utiliza a combinação uniforme parametrizada, proposto em [62], para a operação de cruzamento, onde são combinados dois indivíduos selecionados aleatoriamente de toda a população. Seja n o número de genes dos indivíduos presentes na população. Dentre esses, os indivíduos c_1 e c_2 são escolhidos aleatoriamente na população e p é a probabilidade de um indivíduo descendente c_3 herdar um gene do indivíduo c_1. O novo indivíduo c_3 é gerado da seguinte forma: para $i = 1, \ldots, n$, o seu *i-ésimo* gene $c_3(i)$ recebe a *i-ésima* chave do indivíduo c_1 com uma probabilidade p ou do indivíduo c_2, com probabilidade $1 - p$.

A Figura 1.2 ilustra o processo de cruzamento de dois indivíduos com $n = 4$ e $p = 0,7$. Para cada gene do novo indivíduo, um número real é gerado aleatoriamente no intervalo [0,1). O novo indivíduo herda o gene do indivíduo 1, se o valor gerado for menor ou igual a $0,7$. Caso contrário, ele herda do indivíduo 2. Nesse exemplo, o novo indivíduo se

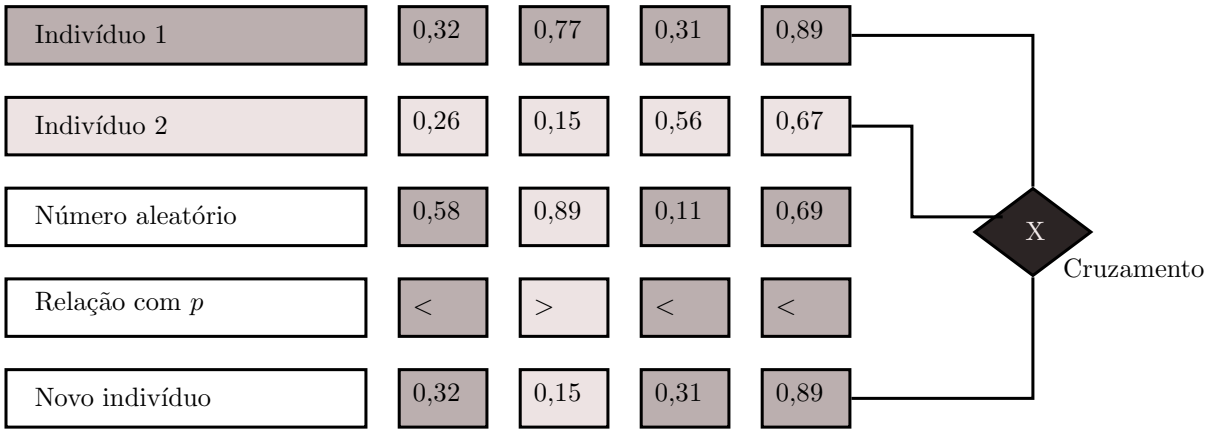


Figura 1.2: Combinação uniforme parametrizada.

assemelha mais ao indivíduo 1, visto que herdou o primeiro, terceiro e quarto genes do último.

Os algoritmos genéticos com chaves aleatórias tendenciosas (do inglês *biased random-key genetic algorithm* – BRKGA) foram primeiramente propostos em [17, 20, 21]. Um BRKGA se difere de um RKGA no modo como os indivíduos são selecionados para o cruzamento (ver Gonçalves e Resende [22] para uma revisão e Resende e Ribeiro [54] para um tutorial). No RKGA, a seleção dos pais para o cruzamento é aleatória em todo o conjunto de indivíduos, enquanto no BRKGA a seleção obrigatoriamente será de um pai do conjunto elite, que contém os indivíduos mais aptos, e o outro do restante da população. Um BRKGA também utilizam a combinação uniforme parametrizada de Spears e DeJong [62].

Um BRKGA evolui uma população de vetores de chaves aleatórias ao longo de um número de gerações. A cada nova geração, a população é dividida em dois subconjuntos: TOP e REST. O tamanho da população é |TOP| + |REST|. O subconjunto TOP contém as melhores soluções presentes na população e o subconjunto REST é formado por dois subconjuntos disjuntos: MID e BOT. As piores soluções da população encontram-se em BOT e o subconjunto MID contém as demais soluções. Como ilustrado na Figura 1.3, as soluções presentes em TOP são copiadas para a população na próxima geração. Os elementos presentes em BOT são substituídos por novas soluções, geradas aleatoriamente, e então inseridos na próxima geração. As |MID| soluções restantes são obtidas pelo cruzamento de uma solução, escolhida aleatoriamente, do subconjunto TOP e de outra selecionada no subconjunto REST.

A adoção da estratégia de *restart* é uma forma de garantir a diversidade da população [19]. Quando uma heurística BRKGA adota a estratégia de restart, após uma

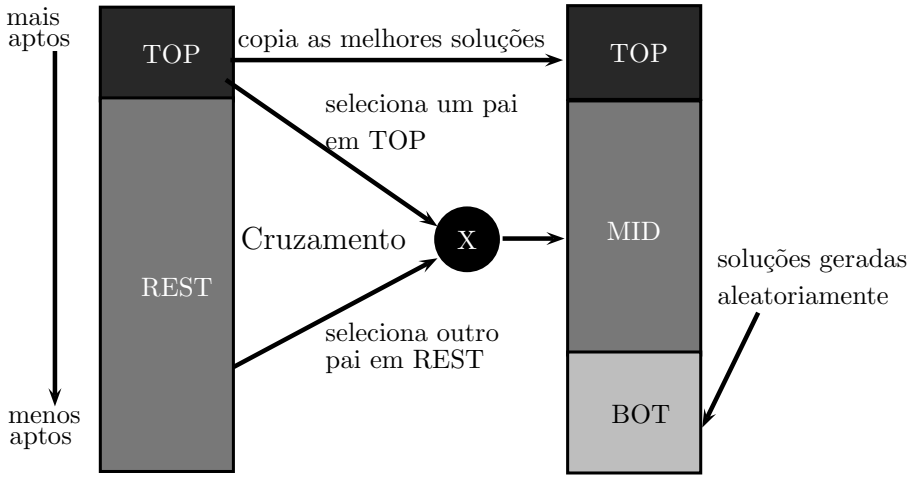


Figura 1.3: Ilustração do processo de transição entre duas gerações.

determinada quantidade de gerações sem melhoria da melhor solução, todos os indivíduos da população atual são substituídos por novos gerados aleatoriamente, com exceção do indivíduo que corresponde à melhor solução.

A Figura 1.4 apresenta o *framework* para o desenvolvimento de algoritmos baseados na metaheurística BRKGA, no qual é necessário desenvolver apenas o decodificador, que é a única parte dependente do problema [63]. O *framework* BRKGA requer a definição dos seguintes parâmetros [23]: (a) o tamanho da população ($p = |TOP| + |REST|$); (b) a fração *pe* da população que corresponde ao conjunto TOP; (c) a fração *pm* da população que corresponde ao conjunto mutante BOT; (d) a probabilidade *rhoe* de selecionar um gene do pai oriundo do conjunto TOP; e (e) o número k de gerações sem melhoria na melhor solução antes de reiniciar a população.

De acordo com Brandão [10], quando a topografia do espaço de soluções é plana, um BRKGA não é significativamente melhor do que o RKGA. O BRKGA fica parecido com o RKGA, necessitando aumentar o valor de *pm* e diminuir o valor de *rhoe*. Um problema com essa característica é o *Steiner Triple Covering* [10].

1.1 Aplicações de BRKGA

Problemas solucionados por heurísticas baseadas em BRKGA obtiveram resultados tão bons ou melhores que aquelas que utilizaram abordagens baseadas em RKGA [19, 23, 54] ou em algoritmos genéticos clássicos [22]. A seguir são apresentados alguns exemplos de sucesso da aplicação de heurísticas baseadas em BRKGA.

Um BRKGA e uma nova busca local para o problema de localização de facilidades

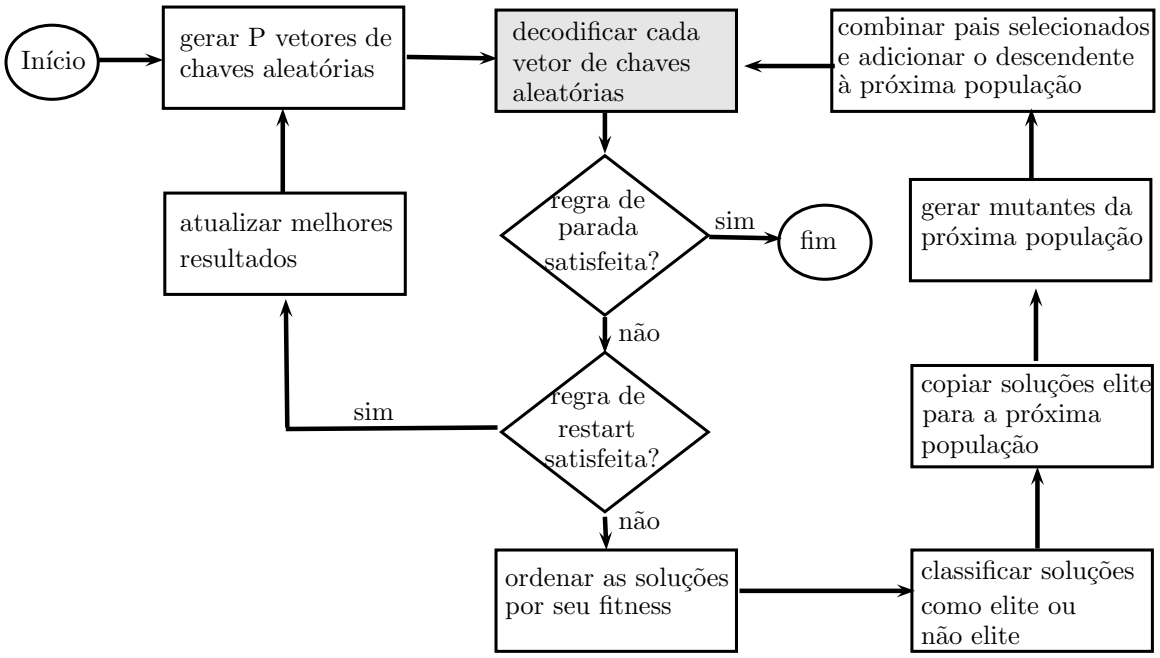


Figura 1.4: Framework BRKGA.

capacitadas em dois níveis (do inglês *two-stage capacitated facility location problem* – TSCFLP) foram propostos por Biajoli et al. em [9]. Nesse problema, um produto deve ser transportado de produtores aos consumidores, passando por depósitos intermediários. O objetivo desse problema consiste em minimizar os custos operacionais, satisfazendo restrições de demanda e capacidade. Experimentos computacionais demonstraram que o BRKGA proposto, quando comparado com o algoritmo estado-da-arte HEA/FA de [24], encontrou 44 melhores valores de soluções dentre as 50 instâncias avaliadas. Em comparação com o algoritmo CS+CPLEX de [52], o BRKGA obteve os melhores valores de solução em 11% das instâncias, além de melhorar a média dos valores das soluções encontradas.

Almeida et al., em [5], investigaram o problema de agendamento de projetos com recursos limitados e flexíveis (do inglês *project scheduling problem with flexible resources* – PSPFR). O problema consiste em realizar o agendamento de um conjunto de atividades que necessitam recursos específicos, de modo a minimizar o *makespan* do projeto. Eles estudaram diferentes métodos na literatura e propuseram um BRKGA para o problema. Os resultados computacionais obtidos indicaram que o BRKGA alcançou melhores resultados quando comparado com a heurística PSS proposta em [4].

Brandão et al., em [11], propuseram um BRKGA para resolver o problema de escalonamento de cargas divisíveis em um único período com processadores dedicados e heterogêneos. Esse problema consiste em selecionar um subconjunto de processadores, definindo a quantidade de carga a ser processada em cada processador e a ordem em que

os fragmentos de carga serão transmitidos para cada um deles. O objetivo é minimizar o *makespan*. Os resultados obtidos em um conjunto de 720 instâncias com até 160 processadores comprovaram que a solução proposta obteve melhores resultados que a melhor heurística da literatura, HeuRet [3]. O BRKGA encontrou soluções ótimas para 413 instâncias (dos 497 casos em que a solução ótima era conhecida). Também foi proposto um novo conjunto de instâncias em que o BRKGA encontrou valores de soluções, em média, 2,38% melhores do que HeuRet. Em [12] foi proposto um BRKGA para resolver uma variante do problema tratado em [11], onde as cargas são divisíveis em múltiplos períodos. O objetivo também foi minimizar o *makespan*. Os resultados obtidos foram 11,68% melhores do que o melhor algoritmo da literatura, CBM [59].

O problema de reconfiguração de redes permite reduzir perdas nas redes de distribuição de energia elétrica. Recursos energéticos distribuídos e inovações associadas a redes inteligentes aumentaram a importância de identificar as melhores topologias de rede. A integração de fontes de energia renováveis com saídas aleatórias variáveis implicam na necessidade de remodelar o problema de reconfiguração da rede e na modelagem de técnicas de solução apropriadas. Cavalheiro et al., em [15], propuseram uma nova formulação para o problema que considerava explicitamente as fontes de energia aleatórias e um BRKGA para solucionar esse problema. Os resultados computacionais obtidos indicaram que o BRKGA alcançou melhores resultados, em menor tempo, quando comparado com um algoritmo genético híbrido proposto em [51].

Em redes de transporte, os locais nos quais os recursos são tratados e despachados são conhecidos como *hubs*. O problema de localização de *hub* (do inglês *Hub Location Problems* – HLP) tem como objetivo localizar um conjunto de *hubs* em uma determinada rede e definir o roteamento dos recursos, de modo que o custo total de atender a todas as demandas seja minimizado. Pessoa et al., em [45], investigaram uma variante desse problema, chamada de problema de localização da árvore de *hubs*, no qual os *hubs* são conectados por meio de uma árvore e a infraestrutura de rede depende de uma árvore de abrangência. Foi proposto um BRKGA para resolvê-lo e os resultados obtidos demonstraram que a heurística proposta é robusta e eficaz para esse problema. Em um conjunto de 58 instâncias de teste, o BRKGA alcançou a melhor solução conhecida na literatura em 48 e foi capaz de melhorar as soluções conhecidas de duas.

Em um problema de roteamento e atribuição de comprimentos de onda em redes óticas WDM o principal objetivo é identificar e atribuir uma rota e um comprimento de onda às requisições existentes. Uma variante desse problema é o max-RWA, onde os caminhos

óticos, cujas as rotas utilizem uma mesma fibra ótica, devem utilizar comprimentos de onda diferentes e a quantidade de requisições atendidas ser maximizada. Brandão et al., em [13], propuseram seis heurísticas construtivas e um algoritmo BRKGA. Três heurísticas foram baseadas em algoritmos para o problema de empacotamento e as demais foram baseadas em soluções para o problema de escalonamento em múltiplos processadores. A heurística SPT foi utilizada como decodificador pelo BRKGA, que foi comparado com uma heurística multi-partida proposta no mesmo trabalho, com um método exato proposto em [36] e com uma busca tabu de [16]. Os experimentos computacionais demonstraram que, quando comparado ao método exato, o desvio médio não foi superior a 4% do limite superior. Na comparação com a heurística multi-partida, o BRKGA encontrou melhores soluções. Para o conjunto de instâncias tratadas pela busca tabu, o BRKGA também obteve, em média, melhores resultados. Além disso, o BRKGA foi a primeira heurística a tratar instâncias com mais de 27 nós.

Noronha et al. [37] propuseram um BRKGA para uma variante do problema de roteamento e atribuição de comprimentos, que tem como objetivo minimizar o número de comprimento de onda utilizados, denominado min-RWA. O BRKGA utilizou a melhor heurística da literatura [61] como algoritmo de decodificação. Ele foi comparado com a heurística busca tabu 2-EDR+TS-PCP [40], assim como com uma variante multi-partida da heurística BFD-RWA [61], a MS-RWA. O BRKGA proposto melhorou os resultados da variante MS-RWA, convergindo para melhores soluções, em média, em 23% menos tempo que MS-RWA. As soluções obtidas por BRKGA apresentaram desvio médio quase 50% menor que o apresentado pela busca tabu.

Outras aplicações de heurísticas baseadas em BRKGA são apresentadas em [19]. As aplicações descritas a seguir são as principais contribuições desta tese.

Três variantes do BRKGA para o problema quasi-clique de cardinalidade máxima (MQCP) foram propostas no decorrer desta tese e publicadas em [50] e submetidas para publicação em [46]. O MQCP consiste em encontrar o maior subconjunto de cardinalidade máxima de vértices, tal que a densidade do grafo induzido é maior ou igual a uma densidade alvo (γ). Em grafos densos, as estratégias baseadas em BRKGA foram comparadas com a heurística estado-da-arte *Restarted Iterated Greedy** (RIG*) proposta em [42]. Já em grafos esparsos, a variante BRKGA-IG* foi comparada com dois algoritmos exatos propostos em [64]. As abordagens baseadas em BRKGA obtiveram resultados melhores que a heurística RIG*, e o BRKGA-IG* foi mais rápido que os algoritmos exatos avaliados. Por fim, uma variante híbrida do BRKGA-IG* foi proposta em [46], o BRKGA-

ExactQClique. Em grafos densos, novos resultados computacionais demonstraram que o BRKGA-ExactQClique obteve melhores resultados quando comparados aos obtidos pelo BRKGA-IG*. O Capítulo 2 define o problema e os resultados alcançados são apresentados nos artigos que compõem os Anexos A e B.

Também nesta tese, foi tratado o problema de minimização do número total de comprimentos de onda necessários para rotear demandas programadas (RWA-SSLD). Esse problema consiste em minimizar o número total de comprimentos de onda necessários para rotear um conjunto de demandas de caminhos óticos, de modo que rotas compartilhando fibras óticas, em um determinado período do tempo, usem comprimentos de onda diferentes. Um BRKGA para esse problema foi proposto no decorrer desta tese e submetido para publicação em [47]. Experimentos computacionais demonstraram que a nova abordagem obteve melhores resultados quando comparados aos obtidos por uma heurística multi-partida construtiva (MDP-RWA-SSLD). Além disto, o BRKGA proposto encontrou soluções que usam aproximadamente 50% do número de comprimentos de onda obtidos pela heurística estado-da-arte MCDF-CR proposta em [34]. O Capítulo 3 define o problema e os resultados alcançados são apresentados no artigo que compõe o Anexo C.

1.2 Estrutura do Trabalho

Esta tese apresenta a aplicação dos algoritmos genéticos com chaves aleatórias tendenciosas para dois problemas de otimização e está organizada como segue: o Capítulo 2 apresenta o problema quasi-clique de cardinalidade máxima, MQCP. O Capítulo 3 define o problema de minimização do número total de comprimentos de onda necessários para rotear demandas de caminhos óticos programadas, RWA-SSLD. Por fim, as conclusões são apresentadas no Capítulo 4.

Capítulo 2

Problema quasi-clique de cardinalidade máxima

Dado um grafo $G = (V, E)$, onde V é o conjunto de vértices e $E \subseteq V \times V$ é o conjunto de arestas presentes no grafo G. G é um grafo completo, se existe uma aresta em E conectando cada par de vértices de V. Um 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 composto pelos vértices V' e pelas arestas $E(V') \subseteq E$. Para qualquer $V' \subseteq V$, o subconjunto $E(V') \subseteq E$ é formado por todas as arestas de E com ambas extremidades em V'. A densidade do grafo G é dada pela razão $dens(G) = |E|/(|V| \times (|V| - 1)/2)$. 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. Esse problema está entre os primeiros problemas que foram provados serem NP-difíceis em [31].

Em problemas reais, geralmente os grafos fornecidos apresentam ruídos que podem indicar a falta ou o excesso de arestas. Tal situação motiva a generalização do problema, onde o objetivo é encontrar um subgrafo quase todo conectado com alta densidade, conhecido como quasi-clique de cardinalidade máxima (MQCP). O MQCP é NP-difícil, uma vez que o problema da clique máxima é um caso especial dele [44]. O problema apresenta várias aplicações, incluindo-se classificação de sequência de moléculas em projetos de mapeamento do genoma humano [14], análise de conjuntos massivos de dados de comunicação obtidos em redes sociais ou gráficos de chamadas telefônicas [1], bem como várias aplicações na área de mineração de dados. Logo, dado um grafo $G = (V, E)$, onde V é o conjunto de vértices e $E \subseteq V \times V$ é o conjunto de arestas presentes no grafo G,

e uma densidade alvo $\gamma \in (0, 1]$, uma γ-clique é qualquer subconjunto $C \subseteq V$ tal que a densidade de $G(C)$ é maior ou igual a γ. Uma γ-clique C é maximal se não existe outra γ-clique C' tal que C esteja estritamente contida em C'. O MQCP tem como objetivo encontrar um subconjunto de cardinalidade máxima $C^* \in V$, tal que a densidade do grafo induzido em G por $C*$ é maior ou igual ao valor alvo γ.

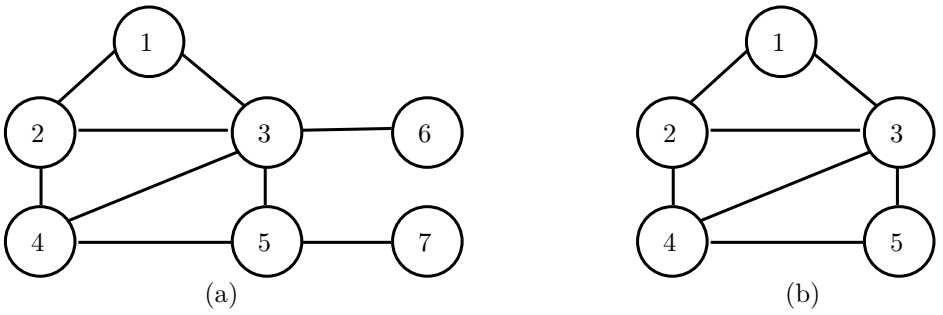


Figura 2.1: (a) Grafo G; (b) Grafo induzido em G por $C*$.

A Tabela 2.1 apresenta exemplos de subconjuntos de vértices do grafo G (Figura 2.1(a)). Dado um valor de $\gamma = 0,78$, apenas os subconjuntos C_2, C_3 e C^* são γ-clique. A melhor solução é o conjunto C^* com cinco vértices. A Figura 2.1(b) apresenta o grafo induzido em G por C^* com $dens(G(C^*)) = 0,70$.

Tabela 2.1: Subconjuntos de vértices de G

	Vértices	Cardinalidade	Densidade do grafo induzido em G
C_1	{1,2,3,4,5,6,7}	7	0,43
C_2	{1,2,3}	3	1,00
C_3	{2,3,4,5}	4	0,83
C_4	{1,2,3,4,5,6}	6	0,53
C^*	**{1,2,3,4,5}**	**5**	**0,70**

2.1 Heurísticas e trabalhos relacionados

Na literatura existem algumas heurísticas para o MQCP baseadas em estratégias bem conhecidas, como algoritmos gulosos aleatórios e suas abordagens reconstrutivas [41, 42], busca local estocástica [14] e GRASP [2]. Veremyev et al. [64] propuseram quatro formulações de programação inteira mista para o problema de MQCP em gráficos esparsos e dois algoritmos baseados nas melhores formulações propostas, que obtiveram melhores resultados que a formulação de programação inteira mista proposta em [44]. Ribeiro e Riveaux [55] desenvolveram um algoritmo exato e enumerativo baseado em propriedades quase-hereditárias e propuseram um novo limite superior, que é usado para podar a árvore de busca e é consistentemente mais restrito que os limites existentes anteriormente.

A heurística construtiva HC3 [42] é uma adaptação da fase construtiva do algoritmo proposto em [2]. Ela cria uma solução inicial, cuja as densidade γ_{temp} é maior ou igual ao limite γ e pode ser decrementada por meio da inserção de novos vértices. A cada iteração, é criada uma lista de vértices candidatos (CL) que podem ser inseridos na solução atual. Uma lista restrita de candidatos (RCL) é construída com os melhores candidatos em CL e um vértice é selecionado aleatoriamente de RCL para ser inserido na solução atual. Esse processo se repete até a lista de candidatos ficar vazia. Um parâmetro α é usado para definir o tamanho da lista restrita de candidatos. Os seguintes critérios são usados, nesta ordem, para selecionar os vértices que serão colocados em CL por HC3 [1, 2, 42]:

1. Grau do vértice: este critério é aplicado somente uma vez. A lista de candidatos CL, na primeira iteração, é formada por todos os vértices. Os vértices que tenham os maiores graus são inseridos em RCL e um deles é selecionado aleatoriamente como o primeiro vértice a fazer parte da solução.

2. Diferença potencial: a lista de candidatos CL é formada por todos os vértices cuja a inserção na solução atual resulta em uma nova solução, cuja a densidade é maior ou igual à densidade atual γ_{temp}. Os vértices em RCL são aqueles com as maiores diferenças de potencial, ou seja, aqueles cuja a inserção aumenta maximamente a densidade da solução atual.

3. Grau para a solução atual: este último critério é aplicado quando não há mais vértice cuja a inserção na solução atual aumente ou mantenha a densidade γ_{temp} da solução atual. A lista de candidatos CL é formada por todos os vizinhos da solução atual. Os vértices em CL com os maiores graus são colocados em RCL. A densidade da solução resultante diminui em relação à densidade corrente γ_{temp} sempre que este critério de seleção é aplicado.

Outras heurísticas construtivas propostas por [42] partem de soluções geradas pela heurística HC3. Elas alternam entre duas fases: destruição parcial da solução atual e reconstrução de uma nova solução viável usando um algoritmo guloso para completar a solução parcialmente destruída. Dentre as variantes dessa abordagem, a estratégia gulosa iterada (IG), usada por [56] em um problema de escalonamento, consiste na aplicação repetida do processo de destruição parcial, seguida pela reconstrução de uma solução viável.

A heurística *Iterated Greedy** (IG*) constrói uma solução inicial usando a heurística construtiva HC3 e aplica repetidamente as fases de destruição e reconstrução da solução.

Dois parâmetros δ e β são usados na fase de destruição: o parâmetro δ controla a fração dos vértices da solução atual que serão removidos, enquanto que o parâmetro β define o tamanho da lista restrita de candidatos (RCL).

A heurística *Restarted Iterated Greedy** (RIG*) aplica repetidamente a heurística IG*, até que a melhor solução encontrada não possa ser melhorada [42]. Além disso, o parâmetro δ é dinamicamente modificado para diversificar a fração da solução que é destruída na fase de destruição de IG*. RIG* superou as demais heurísticas construtivas e reconstrutivas investigadas em [42], em termos de qualidade da solução.

Ribeiro e Riveaux, em [55], propuseram o algoritmo exato e enumerativo QClique, baseado em propriedades quase-hereditárias propostas em [44]. O algoritmo recebe como entrada o limite inferior $LB = \omega^\ell$ e iterativamente procura por um γ-clique de tamanho $k = \omega^{\ell+1}, \omega^{\ell+2}, \ldots$ até falhar. Um novo limite superior foi utilizado para podar a árvore de busca. O limite inferior inicial pode ser definido como $\omega^\ell = 1$, se não houver informações sobre o tamanhos de γ-clique em G, ou calculado por alguma heurística. O maior valor de k, para o qual um γ-clique de tamanho k for encontrado, é o valor ótimo. Experimentos computacionais demonstraram que o algoritmo QClique foi competitivo com as melhores formulações em [44, 64] resolvidas pelo CPLEX e com o algoritmo *branch and bound* proposto em [43], em termos de qualidade da solução e tempo de execução.

2.2 Contribuições

Para resolver o problema MQCP foram propostos três algoritmos genéticos com chaves aleatórias tendenciosas, denominados BRKGA-HCB, BRKGA-IG* e BRKGA-ExactQClique, utilizando os algoritmos Decoder-HCB, Decoder-IG* e Decoder-ExactQClique como decodificadores. Os resultados computacionais obtidos pelos algoritmos BRKGA-HCB e BRKGA-IG* se encontram no artigo que compõe o Anexo A, com os resultados produzidos por estes experimentos disponíveis no Mendeley [49]. Em grafos densos, os algoritmos BRKGA-HCB e o BRKGA-IG* foram comparados com a heurística *Restarted Iterated Greedy** (RIG*) proposta em [42]. Já em grafos esparsos, a variante que apresentou melhor desempenho, BRKGA-IG*, foi comparada com dois algoritmos exatos propostos em [64]. Nos experimentos em grafos densos, o BRKGA-IG* encontrou 97 melhores valores de solução e 96 melhores valores de médias de solução dentre as 100 instâncias avaliadas. O BRKGA-IG* também obteve as soluções que levaram aos menores valores de todas as estatísticas que apresentam os desvios relativos médios dos melhores valores de solução.

Em grafos esparsos, o BRKGA-IG* foi mais rápido do que algoritmos exatos AlgF3 e AlgF4, para todas as instâncias avaliadas, exceto uma.

Uma variante híbrida do BRKGA-IG* foi proposta em [46], o BRKGA-ExactQClique. Nesta variante, o algoritmo QClique, proposto em [55], realiza a busca local no processo de reconstrução do decodificador para melhorar a qualidade das soluções criadas. Esta nova versão do Decoder-IG* foi nomeada de Decoder-ExactQClique. Os resultados computacionais se encontram no artigo que compõe o Anexo B. Em grafos densos, estes resultados demonstraram que a nova variante BRKGA-ExactQClique obteve melhores resultados quando comparados aos obtidos pelo BRKGA-IG*, encontrando 12 valores de soluções não alcançados pelo BRKGA-IG*, enquanto o BRKGA-IG* encontrou apenas dois valores não alcançados pelo BRKGA-ExactQClique. Considerando o valor médio das soluções obtidas, o BRKGA-ExactQClique foi melhor em 28 instâncias, enquanto o BRKGA-IG* foi melhor em 10 instâncias.

Capítulo 3

Problema de minimização do número total de comprimentos de onda necessários para rotear demandas programadas

Segundo Noronha [38], redes óticas do tipo *wavelength division multiplexing* (WDM) permitem o melhor aproveitamento de recursos disponíveis. Essas redes são compostas por canais de comunicação, também chamado de caminhos óticos (do inglês *lightpaths*). Para permitir a transmissão de informação deve-se atribuir um único comprimento de onda para cada caminho ótico. O problema de roteamento e atribuição de comprimentos de onda (do inglês *routing and wavelength assignment* – RWA) consiste em rotear os caminhos óticos e atribuir um comprimento de onda a cada um deles, de modo que os caminhos óticos, cujas as rotas compartilhem alguma fibra ótica, usem comprimentos de onda diferentes. As demandas de caminhos óticos devem ser conhecidas de antemão e não há conversão de comprimento de onda, ou seja, o caminho ótico deve utilizar o mesmo comprimento de onda em todos os enlaces de fibras óticas ao longo do seu percurso. Dois caminhos óticos podem usar o mesmo comprimento de onda desde que não compartilhem nenhum enlace em um determinado instante de tempo.

Existem diversos trabalhos na literatura, diferenciando-se pelos padrões de tráfego, métricas de avaliação utilizadas e pela existência ou não de conversão de comprimentos de onda [7, 13, 32, 37, 39, 40, 53, 66, 67]. O min-RWA [18, 37], variante do problema RWA, consiste em minimizar o número de comprimentos de onda necessários para atender todas as demandas existentes. O problema foi provado ser NP-difícil em [18]. Heurísticas para o min-RWA aparecem em [7, 26, 33, 35, 37, 39, 61].

Esse problema pode considerar modelos de tráfego estático, dinâmico ou agendado.

No caso de modelo agendado, cada demanda necessita usar a rede durante um período de tempo específico. Existem duas variantes desse problema: (a) *scheduled lightpath demands* (RWA-SLD) [32, 60] e (b) *sliding scheduled lightpath demands* (RWA-SSLD) [6, 28, 29, 30, 34, 57, 58], proposto por Wang et al. [65, 66]. De acordo com Kuri et al. [32], os modelos de tráfego agendados são mais realistas devido à natureza periódica do tráfego de dados, que não é homogêneo ao longo do dia.

Kuri et al. [32] definiu o problema RWA-SLD. Seja $G = (V, E)$ um grafo direcionado representando a topologia física da rede, onde V é o conjunto de nós e E representa o conjunto de arcos (cada arco corresponde a um enlace de fibras óticas). Seja também $D = \{d_1, d_2, \ldots, d_m\}$, o conjunto de m demandas de caminhos óticos, onde cada demanda d_i é definida pelo nó origem s_i, nó destino t_i, um número n_i de requisições de caminhos óticos, um tempo de início a_i e um tempo de término b_i, para $i = 1, \ldots, m$. O problema RWA-SLD tem como objetivo a minimização do número total de comprimentos de onda necessários para rotear todas as demandas de caminhos óticos em um período de tempo específico, definido pelos tempos de início e término. Para cada demanda d_i, todos os n_i caminhos óticos devem ser roteados ao longo dos mesmos arcos usando n_i comprimentos de onda diferentes, para $i = 1, \ldots, m$.

Skorin-Kapov [60] propôs heurísticas para resolver o RWA-SLD. Entre elas, a heurística gulosa DP_RWA_SLD baseada no algoritmo para o min-RWA proposto em [35], conforme descrito em detalhes no Anexo C.

3.1 O problema RWA-SSLD

Essa variante do problema, originalmente proposta por Wang et al. [65, 66], é uma generalização do RWA-SLD quando janelas de tempo são definidas para cada demanda por caminhos óticos, em vez de tempos de início e término. A janela de tempo pode ser maior ou igual à duração da demanda. Aplicações reais são mais propensas a se comportar como problemas RWA-SSLD, por exemplo, sempre que uma empresa deseja otimizar o custo de uma rede, definindo o tráfego durante horários com menos uso para operações de rotina, como *backup* de dados [66]. De acordo com Jaekel et al. [30], tais estratégias permitem a otimização energética.

Seja $G = (V, E)$ um grafo direcionado representando a topologia física da rede, onde V é o conjunto de nós e E representa o conjunto de arcos. Seja também $D = \{d_1, d_2, \ldots, d_m\}$, o conjunto de m demandas de caminhos óticos, onde cada demanda d_i é definida pelo nó

origem s_i, nó destino t_i, um número n_i de requisições de caminhos óticos, uma janela de tempo começando em a_i e terminando em b_i, e uma duração τ_i, para $i = 1, \ldots, m$. O tempo de início f_i para a transmissão nos caminhos óticos da demanda d_i não é conhecida previamente, mas deve satisfazer a restrição da janela de tempo $a_i \leq f_i \leq b_i - \tau_i$. A Tabela 3.1 ilustra um exemplo com $m = 4$ demandas para a rede na Figura 3.1.

Tabela 3.1: Exemplo de problema RWA-SSLD com $m = 4$ demandas.

Demandas	s_i	t_i	n_i	a_i	b_i	τ_i
d_1	A	E	2	1	9	4
d_2	B	E	1	3	10	5
d_3	A	D	3	7	12	4
d_4	C	E	4	4	11	3

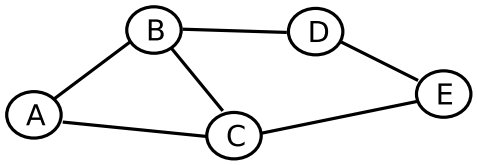


Figura 3.1: Exemplo de topologia de rede.

A Figura 3.2 ilustra uma janela de tempo que começa em a_i e termina em b_i para o agendamento de demanda d_i com duração τ_i iniciando a qualquer momento $f_i \in [a_i, b_i]$.

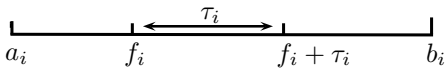


Figura 3.2: Modelo de janela de tempo deslizante.

O exemplo da Figura 3.3 correspondente à demanda d_1 da Tabela 3.1, com o tempo de início $f_1 = 3$ dessa demanda podendo ser antecipado ou atrasado por duas unidades de tempo.

Se todas as janelas de tempo forem iguais a τ_i, isto é, $b_i - a_i = \tau_i$, para $i = 1, \ldots, m$, então não há folgas de tempo e os problemas RWA-SSLD e RWA-SLD coincidem.

3.2 Revisão da literatura

Wang et al. [65, 66] introduziram o problema RWA-SSLD. A abordagem proposta em [66] decompõe o RWA-SSLD em dois subproblemas: (a) posicionar adequadamente uma demanda dentro de sua janela de tempo, de modo a reduzir a sobreposição de tempo entre um conjunto de demandas; e (b) rotear e atribuir comprimentos de onda (RWA) a um

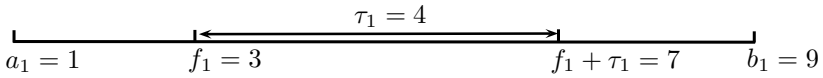


Figura 3.3: Exemplo de uma janela de tempo deslizante para a demanda d_1 da Tabela 3.1.

conjunto de demandas SSLD em redes óticas WDM sem conversão de comprimento de onda. Além disso, outro algoritmo foi usado para reorganizar as demandas que não foram atendidas pelo algoritmo de roteamento e atribuição de comprimentos de onda, buscando um novo tempo de início. O objetivo principal dos algoritmos propostos consistia em realizar o agendamento do conjunto de demandas, de modo a minimizar o número total de comprimentos de onda utilizados em toda a rede, quando os recursos de rede eram suficientes para atender às especificações das demandas. Quando os recursos de rede não eram suficientes para acomodar todas as demandas especificadas, além da minimização do número de demandas que precisam ser reorganizadas, o objetivo também considerava a minimização do número total de comprimentos de onda utilizados, incluindo os comprimentos de onda usados pelas demandas rearranjadas. A rede NSFNET com 14 nós foi usada nos experimentos computacionais, com o número de comprimentos de onda que poderiam ser usados em cada arco limitada a 30 e o número de demandas $m \in [50.400]$. Os experimentos consideraram três classes de demandas: correlação de tempo baixa, média e alta [32].

Liu et al. [34] abordou o caso particular de RWA-SSLD onde $n_i = 1$, para $i = 1, \ldots, m$, usando um modelo de coloração de partição. Os autores também consideraram um limite máximo de δ *hops* para a rota atribuída a uma demanda. As topologias de rede USANET e NSFNET com 24 e 14 nós, respectivamente, foram utilizadas nos experimentos computacionais, com o número de demandas de $m \in [10, 100]$. O algoritmo proposto *maximum conflict degree first conflict reducing* (MCDF-CR) foi comparado com o algoritmo IPSR proposto em [6]. Os resultados demonstraram que, na média, o MCDF-CR reduziu o número de comprimentos de onda usados em 9.9%.

Jaekel et al. [28] propuseram formulações de programação linear inteira para RWA-SLD e RWA-SSLD, minimizando o número de comprimentos de onda utilizados ou o congestionamento da rede (medido em termos do número de comprimentos de onda no arco mais sobrecarregado). A rede NFSNET com 14 nós e a rede proposta em [27] com 10 nós foram usadas nos experimentos computacionais, com o número de comprimentos de onda que podem ser usados em cada arco limitado a 32. Novamente, os experimentos consideraram três classes de demandas: correlação de tempo baixa, média e alta [32]. As janelas de tempo tinham durações de 2, 4 ou 6 horas.

Andrei et al. [6] propuseram uma nova formulação para RWA-SSLD e uma nova heurística nomeada como IPSR-LR (*integrated provisioning of sliding requests*). Eles assumiram que todas as demandas requeriam a mesma quantidade de comprimentos de onda. A rede USANET com 24 nós foi utilizada nos experimentos computacionais, com o número de demandas $m \in [250, 1400]$.

Jaekel et al. [30] investigaram uma variante do RWA-SSLD e RWA-SLD com objetivo de minimizar o consumo energético e o custo de transmissão. Eles propuseram uma nova formulação de programação inteira para os problemas RWA-SLD e RWA-SSLD, bem como um algoritmo genético para instâncias com no máximo 20 nós e centenas de demandas. As redes NSFNET e ARPANET com 14 e 20 nós, respectivamente, foram usadas nos experimentos computacionais, com o número de demandas $m \in [60, 800]$. Novamente seguindo [32], os experimentos incluíram instâncias com três classes de demandas: correlação de tempo baixa, média e alta.

3.3 Contribuições

Nesta tese, o problema RWA-SSLD apresenta recursos de rede ilimitados (capacidade de comprimentos de onda), tendo como objetivo a minimização do número total de comprimentos de onda necessários para rotear todas as demandas dentro das janelas de tempo atribuídas para cada uma delas, usando as notações introduzidas na Seção 3.1. Um algoritmo genético com chaves aleatórias tendenciosas, denominado BRKGA-SSLD, utilizando o algoritmo Decoder-SSLD como decodificador, foi proposto para minimizar o número total de comprimentos de onda necessários para rotear demandas RWA-SSLD.

O estado da arte em problemas RWA-SLD e RWA-SSLD indica que as instâncias são geradas aleatoriamente e não foram disponibilizadas em nenhum repositório [6, 28, 30, 32, 34, 60, 65, 66]. Deste modo, para a geração das demandas RWA-SSLD foi utilizado algoritmo disponibilizado por [32], e utilizado por [60], que é capaz de gerar demandas RWA-SLD respeitando as definições de correlação de tempo. O código disponibilizado pelos autores possui um parâmetro $C \in [0, 1]$ que estabelece a correlação de tempo entre as demandas. Uma correlação de tempo próxima de 0 produz conjuntos de demandas com menor sobreposição no tempo, enquanto uma correlação de tempo próxima de 1 significa que as demandas geradas apresentam elevada sobreposição no tempo. Cada demanda RWA-SLD d gerada foi transformada em uma demanda RWA-SSLD d', a partir da criação de uma janela de tempo por meio dos tempos de início e término a_d e b_d, respectivamente.

A janela de tempo de d' é $[a_{d'}, b_{d'}]$, com $a_{d'} = a_d - \gamma_1(b_d - a_d)$ e $b_{d'} = b_d + \gamma_2(b_d - a_d)$, com γ_1 e γ_2 aleatoriamente gerados no intervalo $[0, B]$. Neste trabalho as instâncias foram geradas com número de demandas $m \in [100, 10000]$, correlação de tempo $C \in [0.600, 0.994]$ e $B \in \{2, 4, 10\}$. As instâncias geradas neste trabalho estão disponíveis no Mendeley [48].

Experimentos computacionais demonstraram que a nova abordagem BRKGA-SSLD obteve melhores resultados quando comparados aos obtidos por uma heurística multi-partida construtiva (MDP-RWA-SSLD). A solução proposta encontrou 59 melhores valores de solução e 58 melhores médias dos valores das soluções encontradas, dentre as 60 instâncias avaliadas. Também obteve as soluções que levaram aos menores valores de todas as estatísticas que apresentam os desvios relativos médios dos melhores valores de solução. Além disto, o BRKGA-SSLD encontrou soluções que usaram aproximadamente 50% do número de comprimentos de onda obtidos pela heurística estado-da-arte MCDF-CR proposta em [34]. O MCDF-CR resolve o caso particular de RWA-SSLD onde $n_i = 1$, para $i = 1, \ldots, m$. Os resultados produzidos pelos experimentos foram disponibilizados no Mendeley [48]. O decodificador e os resultados computacionais alcançados se encontram no artigo que compõe o Anexo C.

Capítulo 4

Conclusões

A principal contribuição desta tese foi demonstrar o desempenho eficaz de versões de algoritmos genéticos com chaves aleatórias tendenciosas para dois diferentes problemas de otimização combinatória: quasi-clique de cardinalidade máxima (MQCP) e minimização do número total de comprimentos de onda necessários para rotear demandas de caminhos óticos programadas (RWA-SSLD).

Primeiramente, foram propostas abordagens baseadas na metaheurística BRKGA para resolver o MQCP. Experimentos computacionais demonstraram que estas novas heurísticas obtiveram resultados superiores aos obtidos pela heurística RIG* [41] para grafos densos. Em relação aos grafos esparsos, os algoritmos BRKGA-IG* e BRKGA-ExactQClique foram mais rápidos que algoritmos exatos encontrados na literatura na busca de valores de solução considerados alvo.

Por fim, o problema RWA-SSLD foi explorado e uma solução utilizando a metaheurística BRKGA foi proposta. Experimentos computacionais demonstraram que a nova abordagem obteve melhores resultados quando comparados àqueles alcançados por uma heurística multi-partida construtiva (MDP-RWA-SSLD) e pela heurística estado-da-arte da literatura (MCDF-CR). Além disto, até o presente momento, as instâncias RWA-SSLD da literatura não foram disponibilizadas em nenhum repositório. Neste trabalho, as instâncias utilizadas em [47] foram geradas baseadas no conceito de correlação de tempo, proposto por Kuri et al. [32], e disponibilizadas no Mendeley [48].

Como trabalho futuro, pretende-se investigar a aplicação da estratégia de múltiplas populações para tentar evitar a estagnação do algoritmo, como sugerido em [19]. Nessa estratégia, após um certo número de gerações, os indivíduos mais adaptados de cada população são copiados para as demais populações, substituindo os piores indivíduos.

O trabalho desenvolvido está apresentado em três artigos nos anexos desta tese. O primeiro artigo (Anexo A) apresenta a revisão da literatura sobre o MQCP e o BRKGA, os decodificadores propostos e os experimentos computacionais realizados. No Anexo B é apresentada uma variante híbrida do BRKGA para o problema MQCP, onde foi incorporado um algoritmo de busca local. O quarto artigo (Anexo C) é dedicado ao problema RWA-SSLD, apresentando a revisão da literatura, a proposta de um decodificador, as novas instâncias geradas e os experimentos computacionais realizados.

Referências

[1] Abello, J. M.; Pardalos, P. M.; Resende, M. G. C. On maximum clique problems in very large graphs. In *External Memory Algorithms* (1999), J. M. Abello and J. S. Vitter, Eds., American Mathematical Society, pp. 119–130.

[2] Abello, J. M.; Resende, M. G. C.; Sudarsky, S. Massive quasi-clique detection. In *Proceedings of the 5th Latin American Symposium on Theoretical Informatics*, J. M. Abello and J. Vitter, Eds., vol. 2286 of *Lecture Notes in Computer Science*. Springer, Berlin, 2002, pp. 598–612.

[3] Abib, E. R.; Ribeiro, C. C. New heuristics and integer programming formulations for scheduling divisible load tasks. In *Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling* (Nashville, 2009), pp. 54–61.

[4] Almeida, B. F.; Correia, I.; da Gama, F. S. Priority-based heuristics for the multi-skill resource constrained project scheduling problem. *Expert Systems with Applications 57* (2016), 91–103.

[5] Almeida, B. F.; Correia, I.; da Gama, F. S. A biased random-key genetic algorithm for the project scheduling problem with flexible resources. *TOP 26* (2018), 283–308.

[6] Andrei, D.; Yen, H. H.; Tornatore, M.; Martel, C. U.; Mukherjee, B. Integrated provisioning of sliding scheduled services over WDM optical networks. *IEEE/OSA Journal of Optical Communications and Networking 1* (2009), 94–105.

[7] Banerjee, D.; Mukherjee, B. A practical approach for routing and wavelength assignment in large wavelength-routed optical networks. *IEEE Journal on Selected Areas in Communications 14* (1996), 903–908.

[8] Bean, J. C. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing 6* (1994), 154–160.

[9] Biajoli, F. L.; Chaves, A. A.; Lorena, L. A. N. A biased random-key genetic algorithm for the two-stage capacitated facility location problem. *Expert Systems with Applications 115* (2019), 418–426.

[10] Brandão, J. S. *Algoritmos Genéticos com Chaves Aleatórias Tendenciosas para Problemas de Otimização em Redes*. Tese de Doutorado, Universidade Federal Fluminense, Niterói, Brasil, 2015.

[11] 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. *International Transactions in Operational Research 22* (2015), 823–839.

[12] 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 24* (2016), 1061–1077.

[13] Brandão, J. S.; Noronha, T. F.; Ribeiro, C. C. A biased random-key genetic algorithm to maximize the number of accepted lightpaths in WDM optical networks. *Journal of Global Optimization 65* (2016), 813–835.

[14] Brunato, M.; Hoos, H.; Battiti, R. On effectively finding maximal quasi-cliques in graphs. In *Learning and Intelligent Optimization: Second International Conference, LION 2007*, V. Maniezzo, R. Battiti, and J.-P. Watson, Eds., vol. 5313 of *Lecture Notes in Computer Science*. Springer, Berlin, 2008, pp. 41–55.

[15] Cavalheiro, E. M. B.; Vergílio, A. H. B.; Lyra, C. Optimal configuration of power distribution networks with variable renewable energy resources. *Computers & Operations Research 96* (2018), 272–280.

[16] Dzongang, C.; Galinier, P.; Pierre, S. A tabu search heuristic for the routing and wavelength assignment problem in optical networks. *Communications Letters, IEEE 9* (2005), 426–428.

[17] Ericsson, M.; Resende, M. G. C.; Pardalos, P. M. A genetic algorithm for the weight setting problem in OSPF routing. *Journal of Combinatorial Optimization 6* (2002), 299–333.

[18] Erlebach, T.; Jansen, K. The complexity of path coloring and call scheduling. *Theoretical Computer Science 255* (2001), 33–50.

[19] Gonçalves, J. F.; Resende, M. G. C. Random-key genetic algorithms. In *Handbook of Heuristics*, R. Martí, P. M. Pardalos, and M. G. C. Resende, Eds. Springer, 2018, pp. 703–715.

[20] Gonçalves, J. F.; Mendes, J. J. M. A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics 8* (2002), 629–642.

[21] Gonçalves, J. F.; Resende, M. G. C. An evolutionary algorithm for manufacturing cell formation. *Computers and Industrial Engineering 47* (2004), 247–273.

[22] Gonçalves, J. F.; Resende, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics 17* (2011), 487–525.

[23] Gonçalves, J. F.; Resende, M. G. C.; Toso, R. F. An experimental comparison of biased and unbiased random-key genetic algorithms. *Pesquisa Operacional 34* (2014), 143–164.

[24] Guo, P.; Cheng, W.; Wang, Y. Hybrid evolutionary algorithm with extreme machine learning fitness function evaluation for two-stage capacitated facility location problems. *Expert Systems with Applications 71* (2017), 57–68.

[25] Holland, J. H. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[26] Hyytia, E.; Virtamo, J. Wavelength assignment and routing in WDM networks. In *Fourteenth Nordic Teletraffic Seminar* (Copenhagen, 1998), pp. 31–40.

[27] Iraschko, R. R.; MacGregor, M. H.; Grover, W. D. Optimal capacity placement for path restoration in STM or ATM mesh-survivable networks. *IEEE/ACM Transactions on Networking 6* (1998), 325–336.

[28] Jaekel, A.; Chen, Y. Demand allocation without wavelength conversion under a sliding scheduled traffic model. In *Fourth International Conference on Broadband Communications, Networks and Systems* (Raleigh, 2007), pp. 495–503.

[29] Jaekel, A.; Chen, Y. Resource provisioning for survivable WDM networks under a sliding scheduled traffic model. *Optical Switching and Networking 6* (2009), 44–54.

[30] Jaekel, A.; Pare, J.; Chen, Y.; Shaabana, A.; Luo, F. Traffic grooming of scheduled demands for minimizing energy consumption. *Photonic Network Communications 29* (2015), 151–163.

[31] Karp, R. M. Reducibility among combinatorial problems. In *Complexity of Computer Computations* (New York, 1972), R. E. Miller and J. W. Thatcher, Eds., Plenum, pp. 85–103.

[32] Kuri, J.; Puech, N.; Gagnaire, M.; Dotaro, E.; Douville, R. Routing and wavelength assignment of scheduled lightpath demands. *IEEE Journal on Selected Areas in Communications 21* (2003), 1231–1240.

[33] Li, G.; Simha, R. The partition coloring problem and its application to wavelength routing and assignment. In *Proceedings of the First Workshop on Optical Networks* (Dallas, 2000).

[34] Liu, Z.; Guo, W.; Shi, Q.; Hu, W.; Xia, M. Sliding scheduled lightpath provisioning by mixed partition coloring in WDM optical networks. *Optical Switching and Networking 10* (2013), 44–53.

[35] Manohar, P.; Manjunath, D.; Shevgaonkar, R. K. Routing and wavelength assignment in optical networks from edge disjoint path algorithms. *IEEE Communications Letters 6* (2002), 211–213.

[36] Martins, A. X. *Metaheurísticas e Formulações para a Resolução do Problema de Roteamento e Alocação de Comprimentos de Onda em Redes Ópticas*. Tese de Doutorado, Universidade Federal de Minas Gerais, Minas Gerais, Brasil, 2011.

[37] Noronha, T.; Resende, M. G. C.; Ribeiro, C. C. A biased random-key genetic algorithm for routing and wavelength assignment. *Journal of Global Optimization 50* (2011), 503–518.

[38] Noronha, T. F. *Algoritmos para Problemas de Otimização Aplicados a Roteamento e Atribuição de Comprimentos de Onda*. Tese de Doutorado, Pontifícia Universidade Católica do Rio de Janeiro, PUC-RIO, Rio de Janeiro, Brasil, 2008.

[39] Noronha, T. F.; Resende, M. G. C.; Ribeiro, C. C. Efficient implementations of heuristics for routing and wavelength assignment. In *Experimental Algorithms*, C. C. McGeoch, Ed., vol. 5038 of *Lecture Notes in Computer Science*. Springer, Berlin, 2008, pp. 169–180.

[40] Noronha, T. F.; Ribeiro, C. C. Routing and wavelength assignment by partition colouring. *European Journal of Operational Research 171* (2006), 797–810.

[41] Oliveira, A. B. Heurísticas para o problema de quasi-clique de cardinalidade máxima. Dissertação de mestrado, Universidade Federal Fluminense, Niterói, Brasil, 2013.

[42] Oliveira, A. B.; Plastino, A.; Ribeiro, C. C. Construction heuristics for the maximum cardinality quasi-clique problem. In *Abstracts of the 10th Metaheuristics International Conference (MIC 2013)* (Singapore, 2013), p. 84.

[43] Pajouh, F. M.; Miao, Z.; Balasundaram, B. A branch-and-bound approach for maximum quasi-cliques. *Annals of Operations Research 216* (2014), 145–161.

[44] Pattillo, J.; Veremyev, A.; Butenko, S.; Boginski, V. On the maximum quasi-clique problem. *Discrete Applied Mathematics 161* (2013), 244–257.

[45] Pessoa, L. S.; Santos, A. C.; Resende, M. G. C. A biased random-key genetic algorithm for the tree of hubs location problem. *Optimization Letters 11* (2017), 1371–1384.

[46] Pinto, B. Q.; Ribeiro, C. C.; Riveaux, J. A.; Rosseti, I. Improving a biased random-key genetic algorithm for the maximum quasi-clique problem with an exact local search. *International Transactions in Operational Research* (2017). Submetido para publicação.

[47] Pinto, B. Q.; Ribeiro, C. C.; Rosseti, I.; Noronha, T. F. A biased random-key genetic algorithm for routing and wavelength assignment under a sliding scheduled traffic model in wdm optical networks. *Information Sciences* (2018). Submetido para publicação.

[48] Pinto, B. Q.; Ribeiro, C. C.; Rosseti, I.; Noronha, T. F. Input data and detailed numerical results for 'A biased random-key genetic algorithm for routing and wavelength assignment under a sliding scheduled traffic model in WDM optical networks'. https://data.mendeley.com/datasets/r76d3pjnnk/2, 2018. Último acesso em outubro de 2018.

[49] Pinto, B. Q.; Ribeiro, C. C.; Rosseti, I.; Plastino, A. Input data and detailed numerical results for 'A biased random-key genetic algorithm for the maximum quasi-clique problem'. https://data.mendeley.com/datasets/khdncrwb5s/1, 2017. Último acesso em outubro de 2018.

[50] Pinto, B. Q.; Ribeiro, C. C.; Rosseti, I.; Plastino, A. A biased random-key genetic algorithm for the maximum quasi-clique problem. *European Journal of Operational Research 271* (2018), 849–865.

[51] Queiroz, L. M. O.; Lyra, C. Adaptive hybrid genetic algorithm for technical loss reduction in distribution networks under variable demands. *IEEE Transactions on Power Systems 24* (2009), 445–453.

[52] Rabello, R. L.; Mauri, G.; Ribeiro, G. M. Método heurístico híbrido para resolução do problema de localização de facilidades capacitadas em dois níveis. In *Anais do xlviii SBPO - Simpósio Brasileiro de Pesquisa Operacional* (Vitória, ES - Brasil, 2016), pp. 2460–2471.

[53] Ramaswami, R.; Sivarajan, K. N. Routing and wavelength assignment in all-optical networks. *IEEE/ACM Transactions on Networking 3* (1995), 489–500.

[54] Resende, M. G. C.; Ribeiro, C. C. Biased-random key genetic algorithms: An advanced tutorial. In *Proceedings of the 2016 Genetic and Evolutionary Computation Conference - GECCO'16 Companion Volume* (Denver, 2016), Association for Computing Machinery, pp. 483–514.

[55] Ribeiro, C. C.; Riveaux, J. A. An exact algorithm for the maximum quasi-clique problem. *Journal of Global Optimization* (2017). Submetido para publicação.

[56] Ruiz, R.; Stützle, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research 177* (2006), 2033–2049.

[57] Saradhi, C.; Gurusamy, M. Scheduling and routing of sliding scheduled lightpath demands in WDM optical networks. In *Conference on Optical Fiber Communication and the National Fiber Optic Engineers Conference* (Anaheim, 2007), pp. 1–3.

[58] Saradhi, C.; Gurusamy, M.; Piesiewicz, R. Routing fault-tolerant sliding scheduled traffic in WDM optical mesh networks. In *5th International Conference on Broadband Communications, Networks and Systems* (2008), pp. 197–202.

[59] Shokripour, A.; Othman, M.; Ibrahim, H.; Subramaniam, S. New method for scheduling heterogeneous multi-installment systems. *Future Generation Computer Systems 28* (2012), 1205–1216.

[60] Skorin-Kapov, N. Heuristic algorithms for the routing and wavelength assignment of scheduled lightpath demands in optical networks. *IEEE Journal on Selected Areas in Communications 24* (2006), 2–15.

[61] Skorin-Kapov, N. Routing and wavelength assigment in optical networks using bin packing based algorithms. *European Journal of Operational Research 177* (2007), 1167–1179.

[62] 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.

[63] Toso, R. F.; Resende, M. G. C. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software 30* (2015), 81–93.

[64] Veremyev, A.; Prokopyev, O. A.; Butenko, S.; Pasiliao, E. L. Exact MIP-based approaches for finding maximum quasi-cliques and dense subgraphs. *Computational Optimization and Applications 64* (2016), 177–214.

[65] Wang, B.; Li, T.; Luo, X.; Fan, Y. Traffic grooming under a sliding scheduled traffic model in WDM optical networks. In *IEEE Workshop on Traffic Grooming in WDM Networks* (San Jose, 2004).

[66] Wang, B.; Li, T.; Luo, X.; Fan, Y.; Xin, C. On service provisioning under a scheduled traffic model in reconfigurable WDM optical networks. In *2nd International Conference on Broadband Networks* (Boston, 2005), pp. 13–22.

[67] Zang, H.; Jue, J. P.; Mukherjeey, B. A review of routing and wavelength assignment approaches for wavelength-routed optical wdm networks. *Optical Networks Magazine 1* (2000), 47–60.

ANEXO A – Pinto, B. Q., Ribeiro, C. C., Rosseti, I., Plastino, A. "A biased random-key genetic algorithm for the maximum quasi-clique problem". European Journal of Operational Research (2018), p. 849 – 865.

European Journal of Operational Research 271 (2018) 849–865

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A biased random-key genetic algorithm for the maximum quasi-clique problem

Bruno Q. Pinto[a,b], Celso C. Ribeiro[b,*], Isabel Rosseti[b], Alexandre Plastino[b]

[a] Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro, Uberlândia, MG 38411-104, Brazil
[b] Institute of Computing, Universidade Federal Fluminense, Niterói, RJ 24210-240, Brazil

ARTICLE INFO

Article history:
Received 26 November 2017
Accepted 30 May 2018
Available online 6 June 2018

Keywords:
Metaheuristics
Biased random-key genetic algorithm
Maximum quasi-clique problem
Maximum clique problem
Graph density

ABSTRACT

Given a graph $G = (V, E)$ and a threshold $\gamma \in (0, 1]$, the maximum cardinality 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. This problem is NP-hard, since it admits the maximum clique problem as a special case. It has a number of applications in data mining, e.g. in social networks or phone call graphs. In this work, we propose a biased random-key genetic algorithm for solving the maximum cardinality quasi-clique problem. Two alternative decoders are implemented for the biased random-key genetic algorithm and the corresponding algorithm variants are evaluated. Computational results show that the newly proposed approaches improve upon other existing heuristics for this problem in the literature. All input data for the test instances and all detailed numerical results are available from Mendeley.

1. Introduction

Let $G = (V, E)$ be a graph defined by a vertex set V and an edge set $E \subseteq V \times V$. 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 $E(V') \subseteq E$ formed by all edges of E with both ends in V'.

The density of graph G is given by $dens(G) = |E|/(|V| \times (|V| - 1)/2)$. The degree $deg_G(v)$ of a node $v \in V$ denotes the number of vertices in G that are adjacent to v.

A graph is complete if there is an edge connecting any pair of its vertices. 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).

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 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 is also NP-hard, since it admits the maximum clique problem as a special case in which $\gamma = 1$, see (Pattillo, Veremyev, Butenko, & Boginski, 2013). 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 (Brunato, Hoos, & Battiti, 2008) and the analysis of massive communication data sets obtained from social networks or phone call graphs (Abello, Pardalos, & Resende, 1999), as well as various data mining and graph mining applications.

A few heuristics for MQCP exist in the literature, based on well known approaches such as greedy randomized algorithms and their iterated extensions (Oliveira, Plastino, & Ribeiro, 2013), stochastic local search (Brunato et al., 2008), and GRASP (Abello, Resende, & Sudarsky, 2002). In this work, we propose two variants of a biased random-key genetic algorithm for solving the maximum quasi-clique problem. The remainder of this article is organized as follows. Section 2 presents the formulation of the maximum quasi-clique problem and reviews exact solution approaches. Heuristics and related work are reviewed in Section 3. Section 4 gives the overall description of biased random-key genetic algorithms and their customization to the maximum quasi-clique problem. Section 5 describes the decoders and two variants of the biased random-key genetic algorithm, each of them based on a different decoder. Computational results are presented in Section 6. Concluding remarks are drawn in the last section. The computational experiments on dense graphs showed that the biased random-key genetic algorithm outperformed the best heuristic in the literature. The experiments on sparse graphs

* Corresponding author.
E-mail addresses: bruno.queiroz@iftm.edu.br (B.Q. Pinto), celso@ic.uff.br (C.C. Ribeiro), rosseti@ic.uff.br (I. Rosseti), plastino@ic.uff.br (A. Plastino).

https://doi.org/10.1016/j.ejor.2018.05.071
0377-2217/

showed that the biased random-key genetic algorithm found results that are competitive with the mixed integer programming approaches in Veremyev, Prokopyev, Butenko, and Pasiliao (2016).

2. Problem formulation

The maximum quasi-clique problem can be formulated by associating a binary variable x_i to each vertex of the graph (Pattillo et al., 2013):

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

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

$$\max \sum_{i \in V} x_i \tag{1}$$

subject to:

$$\sum_{(i,j) \in E : i<j} y_{ij} \geq \gamma \cdot \sum_{i,j \in V : i<j} y_{ij} \tag{2}$$

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

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

$$y_{ij} \geq x_i + x_j - 1, \qquad \forall i, j \in V, \quad i < j, \tag{5}$$

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

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

The objective function (1) maximizes the number of vertices in the solution. If two vertices i, 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 any edge may contribute to the density of a solution only if both of its ends are chosen to belong to this solution. Constraints (5) ensure that any existing edge $(i, j) \in E$ will contribute to the solution if both of its ends are chosen. Constraints (6) and (7) impose the binary and nonnegativity requirements on the problem variables, respectively.

Veremyev et al. (2016) 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 Pattillo et al. (2013), with all mixed integer programs solved using FICO Xpress-Optimizer (FICO, 2017) with the time limit of 3600 seconds. Ribeiro and Riveaux (2018) developed an exact algorithm based on a quasi-hereditary property and 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 approaches in the literature and that their new upper bound is consistently tighter than previously existing bounds.

3. Heuristics and related work

Some heuristics for the maximum quasi-clique problem exist in the literature, based on well known approaches such as greedy randomized algorithms and their iterated extensions (Oliveira et al., 2013), stochastic local search (Brunato et al., 2008), and GRASP (Abello et al., 2002).

The constructive heuristic HC3 (Oliveira et al., 2013) is an adaptation of the construction phase of the algorithm developed by Abello et al. (2002). It builds an initial solution, whose density γ_{temp} is greater than or equal to the threshold γ and may be decreased by the insertion of new vertices. At each iteration, it creates a candidate list of vertices (CL) that can be inserted into the current solution. A restricted candidate list (RCL) is built with the best candidates in CL and a vertex is randomly selected from RCL to be inserted in the current solution, until the candidate list becomes empty. A parameter α is used to define the size of the restricted candidate list. The criteria summarized below are used in this order to select the vertices that will be placed in CL by HC3 (Abello et al., 1999; Abello et al., 2002; Oliveira et al., 2013):

1. Vertex degree: this criterion is applied only once. The candidate list CL at the first iteration is formed by all vertices of the graph. The vertices with the largest degrees are inserted into RCL and one of them is randomly selected as the first vertex to be part of the solution.
2. Potential difference: the candidate list CL is formed by all vertices whose insertion in the current solution results in a new solution whose density is greater than or equal to the current density γ_{temp}. The vertices in RCL are those with the largest potential differences, i.e., those whose insertion increases maximally the density of the current solution.
3. Degree in the current solution: this last criterion is applied when there is no further vertex whose insertion in the current solution increases the density γ_{temp} of the current solution. The candidate list CL is formed by all neighbors of the current solution. The vertices in CL with the largest degrees are placed in RCL. The density of the resulting solution decreases with respect to the current density γ_{temp} whenever this selection criterion is applied.

Other constructive heuristics proposed by Oliveira et al. (2013) start from solutions generated by the greedy randomized heuristic HC3. They alternate between two phases: partial destruction of the current solution and reconstruction of a new feasible solution using a greedy randomized algorithm to complete the partially destroyed solution. Among some variants of this approach, the iterated greedy strategy (IG), used by Ruiz and Stützle (2006) to solve the permutation flowshop scheduling problem, consists in the repeated application of the process of partial destruction, followed by the reconstruction of a feasible solution.

The optimized iterated greedy heuristic (IG*) builds an initial solution using the constructive heuristic HC3 and repeatedly applies a destruction phase followed by a reconstruction phase that applies the HC3 heuristic. Two parameters δ and β are used in the destruction phase: 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 restarted optimized iterated greedy (RIG*) strategy repeatedly applies IG*, until the best solution found can not be improved (Oliveira et al., 2013). Furthermore, parameter δ is dynamically modified to diversify the fraction of the solution that is destroyed in the destruction phase of IG*. This strategy outperformed the others investigated in Oliveira et al. (2013).

4. Biased random-key genetic algorithms for maximum quasi-clique

Genetic algorithms with random keys, or random-key genetic algorithms (denoted by RKGA), were first introduced by Bean (1994) for combinatorial optimization problems whose solutions may be represented by permutation vectors. Solutions are

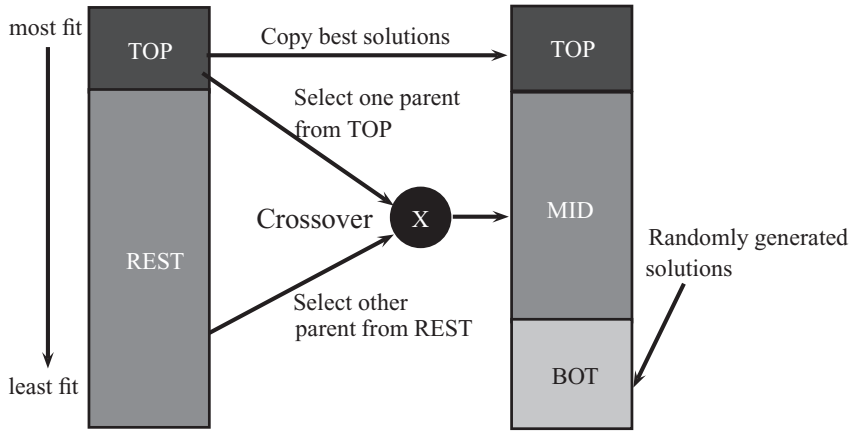


Fig. 1. Population evolution between consecutive generations of a BRKGA.

represented as vectors of randomly generated real numbers called keys. A deterministic algorithm, called a decoder, takes as input a solution vector and associates with it a feasible solution of the combinatorial optimization problem, for which an objective value or fitness can be computed. Two parents are selected at random from the entire population to implement the crossover operation in the implementation of an RKGA. Parents are allowed to be selected for mating more than once in the same generation.

A biased random-key genetic algorithm (BRKGA) differs from an RKGA in the way parents are selected for crossover, see (Gonçalves & Resende, 2011) for a review. In a BRKGA, each element is generated combining one element selected at random from the elite solutions in the current population, while the other is a non-elite solution. The selection is said to be biased because one parent is always an elite solution and has a higher probability of passing its genes to the new generation.

In the following, we propose two variants of a BRKGA for MQCP, each of them using a different decoder. Both of them evolve a population of chromosomes that consists of vectors of real numbers in the interval [0,1) associated with the vertices of the graph G. Each chromosome is decoded by an algorithm that receives the vector of keys and builds a feasible solution for MQCP, i.e., the decoder returns a γ-clique as its output. The two decoders DECODER-HCB and DECODER-IG* will be described in the next section.

We used the parameterized uniform crossover scheme proposed in Spears and de Jong (1991) to combine two parent solutions and to produce an offspring. In this scheme, the offspring inherits each of its keys from the best fit of the two parents with a higher probability. The biased random-key genetic algorithm developed in this work does not make use of the standard mutation operator, where parts of the chromosomes are changed with small probability. Instead, the concept of mutants is used: mutant solutions are introduced in the population in each generation, randomly generated in the same way as in the initial population. 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 (Brandão, Noronha, Resende, & Ribeiro, 2015; 2017; Noronha, Resende, & Ribeiro, 2011).

The keys in the chromosome are randomly generated in the initial population. At each generation, the population is partitioned into two sets: *TOP* and *REST*. The size of the population is $|TOP| + |REST|$. Subset *TOP* contains the best solutions in the population. Subset *REST* is formed by two disjoint subsets: *MID* and *BOT*, with subset *BOT* being formed by the worst elements in the current population. As illustrated in Fig. 1, the chromosomes in *TOP* are simply copied to the population of the next generation. The elements in *BOT* are replaced by newly created mutants that are placed in the new set *BOT*. The remaining elements of the new population are obtained by crossover, with one parent randomly chosen from *TOP* and the other from *REST*. This distinguishes a biased random-key genetic algorithm from the random-key genetic algorithm of Bean (1994), where both parents are selected at random from the entire population. Since a parent solution can be chosen for crossover more than once in a given generation, elite solutions have a higher probability of passing their random keys to the next generation. In this way, $|MID| = |REST| - |BOT|$ offspring solutions are created.

5. Decoders

The implementation of biased random-key genetic algorithms for the maximum quasi-clique problem made use of the C++ library brkgaAPI framework developed by Toso and Resende (2015). The instantiation of the framework shown in Fig. 2 to some specific optimization problem requires exclusively the development of a class implementing the decoder for this problem. This is the only problem-dependent part of the tool. Other applications of this framework in the implementation of biased random-key genetic algorithms can be seen e.g. in Brandão et al. (2015, 2017), Chaves, Lorena, Senne, and Resende (2016), Gonçalves, Resende, and Toso (2014), Ribeiro, Oliveira, Carravilla, and Oliveira (2017), Ruiz, Albareda-Sambola, Fernández, and Resende (2015).

According to Gonçalves et al. (2014), the BRKGA framework requires the following parameters: (a) the population size ($p = |TOP| + |REST|$); (b) the fraction *pe* of the population corresponding to the elite set *TOP*; (c) the fraction *pm* of the population corresponding to the mutant set *BOTTOM*; (d) the probability *rhoe* that the offspring inherits each of its keys from the best fit of the two parents; and (e) the number k of generations without improvement in the best solution until a restart is performed.

We developed two variants of a biased random-key genetic algorithm for solving MQCP: algorithm BRKGA-HCB makes use of the decoder DECODER-HCB based on the HCB constructive heuristic, which is an optimized implementation of the constructive heuristic HC3 (Oliveira et al., 2013), while algorithm BRKGA-IG* makes use of the decoder DECODER-IG* that applies the strategy IG* in the place of HCB. The heuristics and decoders are described next.

5.1. Constructive heuristic HC3

Algorithm 1 describes the constructive heuristic HC3, originally proposed in (Oliveira et al., 2013). It is slighted adapted from the construction phase of the algorithm developed by Abello et al. (2002). It takes as inputs the graph $G = (V, E)$, the threshold γ, and a parameter $\alpha \in [0, 1]$.

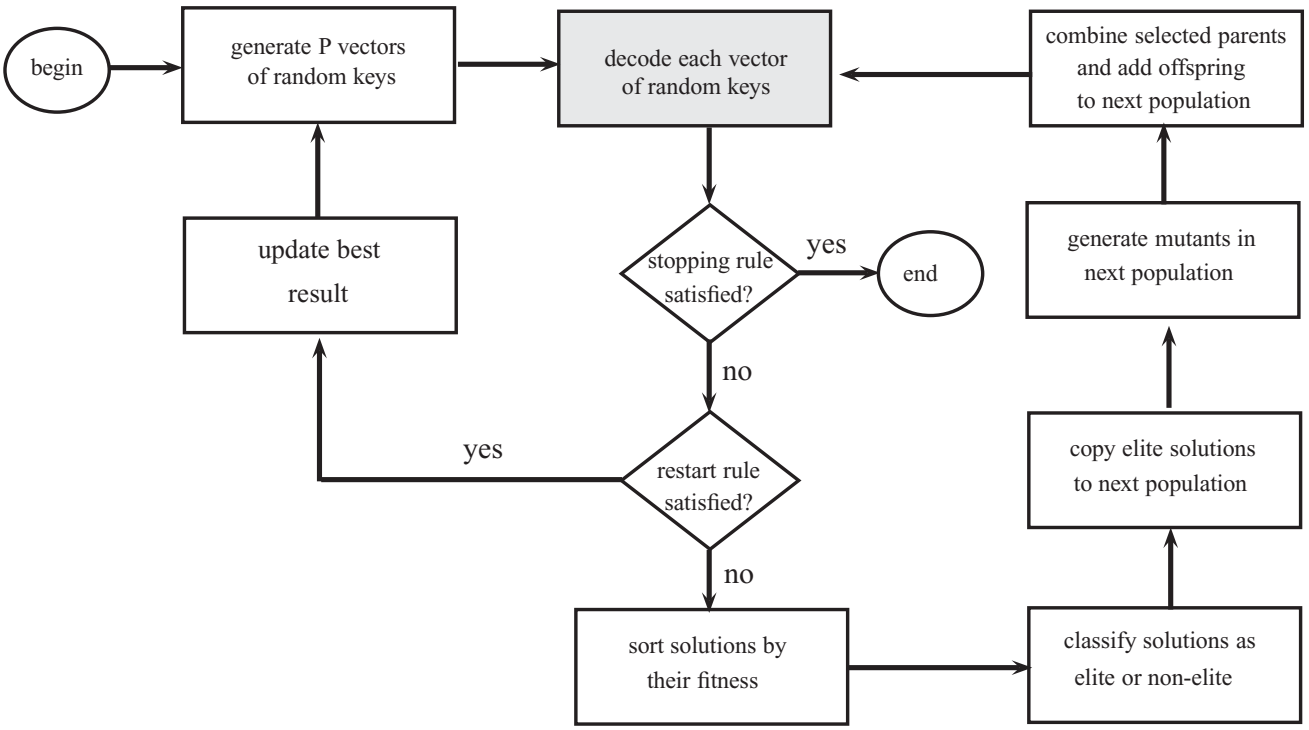


Fig. 2. BRKGA framework.

First, all vertices in V are assigned to the candidate list CL in line 1 and the restricted candidate list RCL is created in line 2, containing the $\max\{1, \alpha \cdot |CL|\}$ vertices with the largest degrees in CL. A vertex x is randomly selected from RCL in line 3 to initialize a solution S_{temp} in line 4. The density γ_{temp} of the graph $G(S_{temp})$ is set to 1 in line 5. The density γ_{temp} of $G(S_{temp})$ will be progressively reduced as new vertices are inserted into S_{temp} along the iterations of the loop in lines 6–34. The temporary solution S_{temp} is copied to S in line 7 and the candidate list CL is reset in line 8. The loop in lines 9–13 places in the candidate list CL the vertices of $V \backslash S$ that may be added to the current solution without reducing the density γ_{temp} of the corresponding γ-clique. If the candidate list CL is not empty, then the potential difference for each candidate vertex is computed in line 16 as proposed in Abello et al. (2002). The restricted candidate list RCL is created in line 18, containing the $\max\{1, \alpha \cdot |CL|\}$ vertices with the largest potential differences in CL. Otherwise, in case the candidate list was empty, the third criterion is applied from lines 19–30. The insertion of any new vertex in the current solution S_{temp} will lead to a reduction in the density γ_{temp} of the graph $G(S_{temp})$. A new candidate solution CL will be built in lines 20–24, containing all neighbors of the current solution S. If this new candidate list is also empty, then the algorithm stops and returns the current solution in line 26, since there are no more candidate vertices to be added to the current solution. Otherwise, the restricted candidate list RCL is created in line 28, containing the $\max\{1, \alpha \cdot |CL|\}$ vertices with the largest degrees in CL. Since the restricted candidate list is not empty, a new vertex $x \in RCL$ is selected to be added to the current solution in line 31. The current solution S_{temp} and its density γ_{temp} are updated in lines 32 and 33, respectively, and a new iteration resumes. The algorithm returns the solution S in line 35.

5.2. Constructive heuristic HCB

The constructive heuristic HCB introduced in this work is a variant of heuristic HC3 discussed in the previous section. While HC3 makes use of a control variable γ_{temp} to avoid that the number of candidates in RCL that satisfy the second criterion (potential differences) becomes very large, it will not be used by HCB since this is relevant only in the case of massive graphs (Abello et al., 2002). We also introduced a minimum size *minsize* for the restricted candidate list, to avoid that it becomes very small for small graphs.

The pseudo-code of Algorithm 2 presents the constructive heuristic HCB, which is basically a simplification of Algorithm 1 considering the two aspects above.

5.3. Decoder DECODER-HCB

Each solution of the maximum quasi-clique problem is associated with a set of $|V|$ random keys. Each random key is a real number in the range [0,1) and corresponds to a vertex of the graph. Each chromosome represented by a set of random keys is decoded by an algorithm (the decoder) that receives the keys and builds a feasible solution to MQCP. In other words, the decoder returns a γ-clique associated with the set of random keys.

Decoder DECODER-HCB to MQCP, whose pseudo-code is given by Algorithm 3, is based on and derived from the constructive heuristic HCB. It is used in two situations, as it will be described later in detail. First, to build a solution from scratch. Second, to complete (i.e., to reconstruct) a partially destroyed solution. In the second case, the decoder receives as an additional parameter a partial solution S formed by a non-empty list of vertices, while in the first case, $S = \emptyset$. DECODER-HCB decodes a population of random keys $r_j \in [0, 1), j = 1, \ldots, |V|$.

5.4. Decoder DECODER-IG*

The second decoder is an extension of DECODER-HCB proposed in the previous section. It is based on the constructive heuristic HCB and on the optimized iterated greedy heuristic IG* described in Section 3, but decodes a population formed by longer vectors of $2 \cdot |V|$ random keys $R_j \in [0, 1), j = 1, \ldots, |V|, |V| + 1, \ldots, 2 \cdot |V|$ each. The first $|V|$ positions of each vector of random keys are used in the construction of the initial solution and in the reconstruction

Algorithm 1 HC3(G, γ, α)

$CL \leftarrow V$
$RCL \leftarrow \{v \in CL : |\{v' \in CL : deg_G(v') \geq deg_G(v)\}| \leq \max\{1, \alpha \cdot |CL|\}\}$
Randomly select $x \in RCL$
$S_{temp} \leftarrow \{x\}$
$\gamma_{temp} \leftarrow 1$
while $\gamma_{temp} \geq \gamma$ **do**
  $S \leftarrow S_{temp}$
  $CL \leftarrow \emptyset$
  for all $v \in V \setminus S$ **do**
    if $\frac{|E(S)|+deg_{G(S)}(v)}{(|S|+1)\cdot|S|/2} \geq \gamma_{temp}$ **then**
      $CL \leftarrow CL \cup \{v\}$
    end if
  end for
  if $CL \neq \emptyset$ **then**
    for all $v \in CL$ **do**
      $dif_v \leftarrow deg_{G(CL)}(v) + |CL| \cdot (deg_{G(S)}(v) - \gamma_{temp} \cdot (|S|+1))$
    end for
    $RCL \leftarrow \{v \in CL : |\{v' \in CL : dif(v') \geq dif(v)\}| \leq \max\{1, \alpha \cdot |CL|\}\}$
  else
    for all $v \in V \setminus S$ **do**
      if $deg_{G(S)}(v) > 0$ **then**
        $CL \leftarrow CL \cup \{v\}$
      end if
    end for
    if $CL = \emptyset$ **then**
      return S
    else
      $RCL \leftarrow \{v \in CL : |\{v' \in CL : deg_{G(S)}(v') \geq deg_{G(S)}(v)\}| \leq \max\{1, \alpha \cdot |CL|\}\}$
    end if
  end if
  Randomly select $x \in RCL$
  $S_{temp} \leftarrow S \cup \{x\}$
  $\gamma_{temp} \leftarrow |E(S_{temp})| / \binom{|S_{temp}|}{2}$
end while
return S

Algorithm 2 HCB(G, γ, α, $minsize$)

$CL \leftarrow V$
$RCL \leftarrow \{v \in CL : |\{v' \in CL : deg_G(v') \geq deg_G(v)\}| \leq \max\{minsize, \alpha \cdot |CL|\}\}$
Randomly select $x \in RCL$
$S \leftarrow \{x\}$
while $CL \neq \emptyset$ **do**
  $CL \leftarrow \emptyset$
  for all $v \in V \setminus S$ **do**
    if $\frac{|E(S)|+deg_{G(S)}(v)}{(|S|+1)\cdot|S|/2} \geq \gamma$ **then**
      $CL \leftarrow CL \cup \{v\}$
    end if
  end for
  if $CL \neq \emptyset$ **then**
    for all $v \in CL$ **do**
      $dif_v \leftarrow deg_{G(CL)}(v) + |CL| \cdot (deg_{G(S)}(v) - \gamma \cdot (|S|+1))$
    end for
    $RCL \leftarrow \{v \in CL : |\{v' \in CL : dif(v') \geq dif(v)\}| \leq \max\{minsize, \alpha \cdot |CL|\}\}$
    Randomly select $x \in RCL$
    $S \leftarrow S \cup \{x\}$
  end if
end while
return S

Algorithm 3 DECODER-HCB(G, γ, α, $minsize$, S, r)

$CL \leftarrow V \setminus S$
if $S = \emptyset$ **then**
  $RCL \leftarrow \{v \in CL : |\{v' \in CL : deg_G(v') \geq deg_G(v)\}| \leq \max\{minsize, \alpha \cdot |CL|\}\}$
  $x \leftarrow argmin\{r_j : j \in RCL\}$
  $S \leftarrow \{x\}$
end if
while $CL \neq \emptyset$ **do**
  $CL \leftarrow \emptyset$
  for all $v \in V \setminus S$ **do**
    if $\frac{|E(S)|+deg_{G(S)}(v)}{(|S|+1)\cdot|S|/2} \geq \gamma$ **then**
      $CL \leftarrow CL \cup \{v\}$
    end if
  end for
  if $CL \neq \emptyset$ **then**
    for all $v \in CL$ **do**
      $dif_v \leftarrow deg_{G(CL)}(v) + |CL| \cdot (deg_{G(S)}(v) - \gamma \cdot (|S|+1))$
    end for
    $RCL \leftarrow \{v \in CL : |\{v' \in CL : dif(v') \geq dif(v)\}| \leq \max\{minsize, \alpha \cdot |CL|\}\}$
    $x \leftarrow argmin\{r_j : j \in RCL\}$
    $S \leftarrow S \cup \{x\}$
  end if
end while
return S

phase of the IG* strategy, while the the last $|V|$ positions of each vector of random keys are used in the destruction phase. The roles of parameters α, δ, and β are the same explained in Section 3 for strategy IG*.

Algorithm 4 starts by creating an initial solution S' in line

Algorithm 4 DECODER-IG*(G, γ, α, δ, β, $minsize$, R)

1: $S' \leftarrow$ DECODER-HCB($G, \gamma, \alpha, minsize, \emptyset, R_j : j = 1, \ldots, |V|$)
2: **repeat**
3: $S \leftarrow S'$
4: **for** $k = 1$ to $\delta \cdot |S'|$ **do**
5: $RCL \leftarrow \{v \in S' : |\{v' \in S' : deg_{G(S')}(v') \leq deg_{G(S')}(v)\}| \leq \max\{minsize, \beta \cdot |S'|\}\}$
6: $x \leftarrow argmin\{R_{|V|+j} : j \in RCL\}$
7: $S' \leftarrow S' \setminus \{x\}$
8: **end for**
9: $S' \leftarrow$ DECODER-HCB($G, \gamma, \alpha, minsize, S', R_j : j = 1, \ldots, |V|$)
10: **until** $|S'| \leq |S|$ **or** graph $G(S')$ is not connected
11: **return** S

1, using the decoder DECODER-HCB and the first random keys R_j, $j = 1, \ldots, |V|$. This solution is copied to S in line 3. The loop in lines 2–10 repeats the partial destruction (vertex eliminations) followed by the reconstruction (vertex insertions) of the current solution, until no further improvements can be obtained.

The loop in lines 4–8 removes one by one the $\delta \cdot |S'|$ vertices that should be eliminated from the current solution. A restricted candidate list RCL of size $\max\{minsize, \beta \cdot |S'|\}$ 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 is performed in line 9, where the current, partial solution S' is rebuilt by decoder DECODER-HCB, once again using the first random keys R_j, $j = 1, \ldots, |V|$. The decoder stops when the new solution obtained by destruction-

reconstruction does not improve the incumbent S. The best solution S is returned in line 11.

6. Computational results

In this section, we address the effectiveness of the heuristics based on biased random-key genetic algorithms. We compare the results obtained with the two proposed BRKGA variants with those obtained by the original RIG* implementation of Oliveira et al. (2013) and by the BRKGA algorithm originally presented in Pinto, Plastino, Ribeiro, and Rosseti (2015), which is a preliminary version of BRKGA-HCB. The first proposed variant is called BRKGA-HCB and makes use of the decoder DECODER-HCB described in Section 5.3, while the second one is denoted BRKGA-IG* and makes use of decoder DECODER-IG* proposed in Section 5.4. A restart strategy is incorporated into BRKGA-IG* and we show that it further improves the performance of the heuristic. We also report numerical experiments on sparse graph instances comparing BRKGA-IG* with the mixed integer programming approaches AlgF3 and AlgF4 of Veremyev et al. (2016).

Both algorithms BRKGA-HCB and BRKGA-IG* were implemented in C++ with the GNU GCC compiler C/C++ version 5.4.0. The experiments have been performed on a Lenovo i7-6500U computer with a 2.50 GHz CPU (maximum turbo frequency with 3.10 GHz) with 8 GB of RAM under the operating system Linux Ubuntu 16.04 LTS with parallel processing features disabled.

The numerical experiments on dense graphs involved 67 maximum clique instances of the Second DIMACS Implementation Challenge (DIMACS, 2016; Johnson & Trick, 1996) and 33 maximum clique instances of the Benchmarks with Hidden Optimum Solutions for Graph Problems (BHOSLIB, 2014; Pullan, Mascia, & Brunato, 2011). The experiments on sparse graphs considered 12 instances obtained from the University of Florida Sparse Matrix Collection (Davis & Hu, 2011) that have also been used by Veremyev et al. (2016).

All the input data for the above test instances are available in Mendeley (see (Pinto, Ribeiro, Rosseti, & Plastino, 2017)), together with the forthcoming detailed numerical results that will be reported along this section.

6.1. Tuning

The best parameters for algorithms BRKGA-HCB and BRKGA-IG* have been determined using the IRACE (Alfaro-Fernández, Ruiz, Pagnozzi, & Stützle, 2017; López-Ibánez, Dubois-Lacoste, Stützle, & Birattari, 2011; Pérez Cáceres, López-Ibáñez, & Stützle, 2014) automatic tuning tool. In the first step of the tuning experiment, we determined the best values for parameters α and *minsize* used by the constructive heuristic HCB. In the second step, we looked for the best values for parameters δ and β used by algorithm BRKGA-IG*. Finally, in the third step, we sought the best values for parameters p (population size), pe (fraction of the population corresponding to the elite set), pm (fraction of the population corresponding to the mutant set), and *rhoe* (probability that the offspring inherits each of its keys from the best fit of the two parents) used by the biased random-key genetic algorithm, with the running times limited to one hour and considering the value ranges suggested by Gonçalves and Resende (2011).

The IRACE tuning experiment performed 1000 runs (Bouamama & Blum, 2017; Maschler, Hackl, Riedler, & Raidl, 2017) of each algorithm for 20 additional problem instances selected from different classes, as listed in Table 1: 12 instances from the Second DIMACS Implementation Challenge and eight BHOSLIB instances.

The value ranges considered by IRACE and the best parameter values identified by the tuning experiment are reported in Table 2.

Table 1
Test instances (20) used in the tuning experiment with IRACE for dense graphs.

Instance	$\|V\|$	$\|E\|$	Threshold γ
brock200_4	200	13,089	0.80
brock400_4	400	59,765	0.80
brock800_4	800	207,643	0.80
C2000.5	2000	999,836	0.80
frb30-15-3	450	83,216	0.95
frb35-17-3	595	148,784	0.95
frb40-19-3	760	247,325	0.95
frb45-21-3	945	387,795	0.95
frb50-23-3	1150	579,607	0.95
frb53-24-3	1272	714,229	0.95
frb56-25-3	1400	869,921	0.95
frb59-26-3	1534	1,049,729	0.95
gen200_p0.9_55	200	17,910	0.99
gen400_p0.9_75	400	71,820	0.99
p_hat300-3	300	33,390	0.95
p_hat500-3	500	93,800	0.95
p_hat700-3	700	183,010	0.95
p_hat1500-1	1500	284,923	0.95
p_hat1500-3	1500	847,244	0.95
san400_0.9_1	400	71,820	0.99

Table 2
Value ranges used by IRACE and best parameter values obtained after tuning.

Parameter	Value ranges	BRKGA-HCB	BRKGA-IG*
α	0.01, 0.02, ..., 0.20	0.01	0.01
minsize	1, 2, 3, 4, 5, 6	3	3
β	0.01, 0.02, ..., 0.20	–	0.02
δ	0.01, 0.02, ..., 0.50	–	0.40
p	50, 51, ..., 100	64	91
pe	0.10, 0.11, ..., 0.25	0.22	0.13
pm	0.10, 0.11, ..., 0.30	0.15	0.22
rhoe	0.50, 0.51, ..., 0.80	0.63	0.78

6.2. Experiments on dense graphs

We considered 100 test problems for the comparative evaluation of the two biased random-key genetic algorithms BRKGA-HCB and BRKGA-IG* proposed in this work with the optimized restarted iterated greedy strategy RIG* (Oliveira et al., 2013) and with the preliminary BRKGA algorithm in Pinto et al. (2015). None of these instances was used in the tuning experiments reported in Section 6.1. Each algorithm was run ten independent times for each instance using different seeds. Algorithm RIG* was made to stop after 100 iterations without improvement in the incumbent. The average time taken by algorithm RIG* over the ten runs for each problem was used as the stopping criterion for each of the BRKGA variants. Therefore, all algorithms are subject to exactly the same stopping criterion.

Tables 3–6 display the number of nodes $|V|$, the number of edges $|E|$, the density, and the threshold γ for each instance. In addition, for each instance and for each algorithm, these tables show the best and the average solution values over the ten runs. The last column in each table shows the running time in seconds observed for RIG*, which was used as the stopping criterion for the BRKGA variants. Cells highlighted in boldface indicate the algorithms that attained the best values for each heuristic. These tables show that the biased random-key genetic algorithm variants BRKGA-HCB and BRKGA-IG* found systematically better solutions than RIG*. In fact, while for only two (MANN_a27 and MANN_a45 – both of them being very dense graphs with density greater than 99%) out of the 100 instances RIG* found a solution that was not matched by at least one of the genetic algorithm variants, either

Table 3
Results for algorithms BRKGA, BRKGA-HCB, BRKGA-IG*, and RIG* – Part I.

					BRKGA		BRKGA-HCB		BRKGA-IG*		RIG*		Running
Instance	$\|V\|$	$\|E\|$	dens. (%)	γ	Best	Average	Best	Average	Best	Average	Best	Average	time (seconds)
C125.9	125	6963	89.85	0.999	**34**	**34.00**	**34**	**34.00**	**34**	**34.00**	**34**	**34.00**	1.82
C250.9	250	27,984	89.91	0.999	**44**	**44.00**	**44**	**44.00**	**44**	**44.00**	**44**	43.30	5.29
C500.9	500	112,332	90.05	0.999	**57**	55.80	**57**	56.20	**57**	**56.70**	55	54.00	12.09
C1000.9	1000	450,079	90.11	0.999	66	65.10	**67**	65.00	**67**	**65.80**	64	62.40	43.61
C2000.9	2000	1,799,532	90.02	0.999	73	71.60	72	71.30	**74**	**72.20**	71	68.70	95.67
C4000.5	4000	4,000,268	50.02	0.800	41	40.60	40	39.30	**44**	**42.60**	42	38.40	69.01
DSJC500.5	500	62,624	50.20	0.800	33	32.20	**34**	32.40	**34**	**32.80**	31	30.20	5.93
DSJC1000.5	1000	249,826	50.02	0.800	37	35.90	**38**	35.00	37	**36.20**	35	33.70	13.34
MANN_a9	45	918	92.73	0.999	**16**	**16.00**	**16**	**16.00**	**16**	**16.00**	**16**	**16.00**	0.24
MANN_a27	378	70,551	99.01	0.999	133	132.20	133	132.40	133	132.40	**135**	**134.60**	37.21
MANN_a45	1035	533,115	99.63	0.999	426	425.10	423	422.20	425	421.20	**439**	**437.60**	640.09
brock200_1	200	14,834	74.54	0.800	**114**	113.30	**114**	113.70	**114**	**114.00**	**114**	113.00	8.17
brock200_2	200	9876	49.63	0.800	**24**	23.20	**24**	**24.00**	**24**	**24.00**	23	22.40	1.44
brock200_3	200	12,048	60.54	0.800	**41**	40.30	**41**	**40.70**	**41**	**40.70**	40	39.20	2.70
brock400_1	400	59,723	74.84	0.800	**189**	187.60	**189**	187.90	**189**	**189.00**	188	186.60	36.48
brock400_2	400	59,786	74.92	0.800	185	184.40	**186**	184.80	**186**	**185.90**	185	183.20	32.45
brock400_3	400	59,681	74.79	0.800	186	185.50	**187**	185.90	**187**	**186.10**	186	184.80	36.29
brock800_1	800	207,505	64.93	0.800	92	90.70	92	89.50	**96**	**93.30**	87	85.10	31.84
brock800_2	800	208,166	65.13	0.800	92	90.00	90	87.80	**93**	**91.70**	87	85.80	31.75
brock800_3	800	207,333	64.87	0.800	90	89.40	90	88.30	**92**	**90.90**	87	84.50	26.73
c-fat200-1	200	1534	7.71	0.500	**30**	**30.00**	**30**	**30.00**	**30**	**30.00**	**30**	29.80	0.42
c-fat200-2	200	3235	16.26	0.500	**58**	**58.00**	**58**	**58.00**	**58**	**58.00**	**58**	**58.00**	0.92
c-fat200-5	200	8473	42.58	0.500	**148**	**148.00**	**148**	**148.00**	**148**	**148.00**	**148**	**148.00**	5.04
c-fat500-1	500	4459	3.57	0.500	**35**	**35.00**	**35**	**35.00**	**35**	**35.00**	**35**	34.20	0.66
c-fat500-2	500	9239	7.33	0.500	**66**	**66.00**	**66**	**66.00**	**66**	**66.00**	**66**	65.40	1.72
c-fat500-5	500	23,191	18.59	0.500	**164**	**164.00**	**164**	**164.00**	**164**	**164.00**	**164**	163.50	7.86
c-fat500-10	500	46,627	37.38	0.500	**324**	**324.00**	**324**	**324.00**	**324**	**324.00**	**324**	**324.00**	27.26
frb30-15-1	450	83,198	82.35	0.950	57	56.30	**59**	57.20	**59**	**58.40**	56	54.70	11.39

Table 4
Results for algorithms BRKGA, BRKGA-HCB, BRKGA-IG*, and RIG* – Part II.

					BRKGA		BRKGA-HCB		BRKGA-IG*		RIG*		Running
Instance	$\|V\|$	$\|E\|$	dens. (%)	γ	Best	Average	Best	Average	Best	Average	Best	Average	time (seconds)
frb30-15-2	450	83151	82.31	0.950	57	56.20	57	56.00	**58**	**56.90**	55	53.60	10.82
frb30-15-4	450	83194	82.35	0.950	56	55.70	58	55.70	**60**	**57.20**	55	52.80	11.13
frb30-15-5	450	83231	82.39	0.950	58	56.70	58	56.90	**59**	**58.60**	56	54.20	12.32
frb35-17-1	595	148,859	84.24	0.950	75	74.10	75	73.90	**78**	**76.60**	73	70.20	23.07
frb35-17-2	595	148,868	84.24	0.950	72	70.90	**74**	70.50	**74**	**73.40**	70	68.40	20.95
frb35-17-4	595	148,873	84.24	0.950	77	75.80	79	77.20	**80**	**78.60**	74	72.80	23.21
frb35-17-5	595	148,572	84.07	0.950	77	75.10	78	75.50	**79**	**77.60**	73	71.30	24.84
frb40-19-1	760	247,106	85.68	0.950	107	105.10	108	105.60	**111**	**109.70**	102	100.30	44.72
frb40-19-2	760	247,157	85.69	0.950	98	96.40	98	95.50	**102**	**100.40**	96	92.00	41.34
frb40-19-4	760	246,815	85.57	0.950	94	92.00	92	90.70	**95**	**94.00**	91	87.20	43.53
frb40-19-5	760	246,801	85.57	0.950	98	96.00	96	94.10	**99**	**97.50**	92	89.70	43.92
frb45-21-1	945	386,854	86.73	0.950	117	115.30	117	114.90	**121**	**120.00**	114	111.40	64.34
frb45-21-2	945	387,416	86.86	0.950	114	112.60	114	111.30	**120**	**118.50**	112	108.30	56.12
frb45-21-4	945	387,491	86.87	0.950	123	122.20	123	119.80	**128**	**126.00**	119	116.00	74.75
frb45-21-5	945	387,461	86.87	0.950	116	113.70	114	112.30	**121**	**118.40**	111	109.20	70.98
frb50-23-1	1150	580,603	87.88	0.950	152	148.80	148	146.20	**158**	**154.80**	146	143.40	123.42
frb50-23-2	1150	579,824	87.76	0.950	152	147.70	151	145.80	**154**	**153.10**	147	144.00	107.99
frb50-23-4	1150	580,417	87.85	0.950	149	145.60	145	142.70	**155**	**152.80**	143	141.40	103.64
frb50-23-5	1150	580,640	87.89	0.950	152	149.20	149	146.10	**158**	**155.20**	149	144.80	117.98
frb53-24-1	1272	714,129	88.34	0.950	190	186.10	186	183.10	**199**	**196.50**	191	184.70	159.12
frb53-24-2	1272	714,067	88.34	0.950	162	160.30	160	157.90	**169**	**167.10**	159	157.10	136.73
frb53-24-4	1272	714,048	88.33	0.950	172	167.90	169	165.60	**178**	**176.50**	170	163.90	143.50
frb53-24-5	1272	714,130	88.34	0.950	158	157.00	155	153.10	**167**	**164.50**	158	153.50	120.71
frb56-25-1	1400	869,624	88.80	0.950	212	208.90	210	207.90	**225**	**223.10**	216	211.00	217.61
frb56-25-2	1400	869,899	88.83	0.950	200	198.50	201	196.20	**212**	**209.30**	200	196.50	224.00
frb56-25-4	1400	869,262	88.76	0.950	187	183.60	182	178.40	**194**	**192.20**	183	177.40	182.62
frb56-25-5	1400	869,699	88.81	0.950	189	186.40	187	184.00	**201**	**197.90**	187	184.30	192.85
frb59-26-1	1534	1,049,256	89.24	0.950	227	223.60	225	219.90	**239**	**237.40**	231	224.30	264.45

Table 5
Results for algorithms BRKGA, BRKGA-HCB, BRKGA-IG*, and RIG* – Part III.

					BRKGA		BRKGA-HCB		BRKGA-IG*		RIG*		Running
Instance	$\|V\|$	$\|E\|$	Dens. (%)	γ	Best	Average	Best	Average	Best	Average	Best	Average	time (seconds)
frb59-26-2	1534	1,049,648	89.27	0.950	226	221.70	220	216.90	**236**	**232.70**	226	220.60	234.40
frb59-26-4	1534	1,048,800	89.20	0.950	216	214.30	216	211.10	**227**	**225.60**	218	213.80	251.09
frb59-26-5	1534	1,049,829	89.29	0.950	210	207.90	205	203.40	**224**	**219.70**	210	204.70	198.77
frb100-40	4000	7,425,226	92.84	0.950	1819	1815.60	1819	1817.30	**1837**	1835.50	**1837**	**1836.00**	6530.86
gen200_p0.9_44	200	17,910	90.00	0.999	40	40.00	40	40.00	**42**	**40.40**	40	39.80	3.91
gen400_p0.9_55	400	71,820	90.00	0.999	52	51.40	**53**	52.20	**53**	**52.40**	51	50.70	10.47
gen400_p0.9_65	400	71,820	90.00	0.999	52	50.80	55	53.20	**66**	**62.00**	51	49.20	9.24
hamming6-2	64	1824	90.48	0.950	**37**	**37.00**	**37**	**37.00**	**37**	**37.00**	**37**	**37.00**	0.64
hamming6-4	64	704	34.92	0.500	**32**	**32.00**	**32**	**32.00**	**32**	**32.00**	**32**	**32.00**	0.44
hamming8-2	256	31,616	96.86	0.999	**129**	**129.00**	**129**	**129.00**	**129**	**129.00**	**129**	**129.00**	13.18
hamming8-4	256	20,864	63.92	0.800	**71**	**71.00**	**71**	**71.00**	**71**	**71.00**	**71**	**71.00**	4.89
hamming10-2	1024	518,656	99.02	0.999	**525**	**525.00**	**525**	**525.00**	**525**	**525.00**	**525**	**525.00**	505.75
hamming10-4	1024	434,176	82.89	0.950	**82**	**81.60**	81	80.70	**82**	81.20	81	79.00	38.48
johnson8-2-4	28	210	55.56	0.800	**5**	**5.00**	**5**	**5.00**	**5**	**5.00**	**5**	**5.00**	0.06
johnson8-4-4	70	1855	76.81	0.800	**43**	42.80	**43**	**43.00**	**43**	**43.00**	**43**	**43.00**	1.03
johnson16-2-4	120	5460	76.47	0.800	**34**	**34.00**	**34**	**34.00**	**34**	**34.00**	**34**	**34.00**	1.12
johnson32-2-4	496	107,880	87.88	0.950	**21**	**21.00**	**21**	**21.00**	**21**	**21.00**	**21**	**21.00**	4.59
keller4	171	9435	64.91	0.800	51	51.00	**54**	52.40	**54**	**54.00**	**54**	52.70	3.86
keller5	776	225,990	75.15	0.800	**486**	**486.00**	**486**	**486.00**	**486**	**486.00**	**486**	**486.00**	110.42
keller6	3361	4,619,898	81.82	0.950	269	263.50	267	260.70	**279**	**275.80**	270	263.20	742.24
p_hat300-1	300	10,933	24.38	0.500	**64**	63.10	**64**	63.70	**64**	**64.00**	63	62.30	8.05
p_hat300-2	300	21,928	48.89	0.800	**114**	**114.00**	**114**	**114.00**	**114**	**114.00**	**114**	**114.00**	9.72
p_hat500-1	500	31,569	25.31	0.500	**96**	95.80	**96**	**96.00**	**96**	**96.00**	95	94.60	20.39
p_hat500-2	500	62,946	50.46	0.800	**211**	**211.00**	**211**	**211.00**	**211**	**211.00**	**211**	**211.00**	28.74
p_hat700-1	700	60,999	24.93	0.500	**119**	117.90	118	117.50	**119**	**118.90**	118	116.40	36.10
p_hat700-2	700	121,728	49.76	0.800	**288**	**288.00**	**288**	**288.00**	**288**	**288.00**	**288**	**288.00**	55.17
p_hat1000-1	1000	122,253	24.48	0.500	144	143.20	144	142.20	**145**	**144.10**	142	141.10	60.64
p_hat1000-2	1000	244,799	49.01	0.800	**385**	**385.00**	**385**	**385.00**	**385**	**385.00**	384	384.00	107.91

Table 6
Results for algorithms BRKGA, BRKGA-HCB, BRKGA-IG*, and RIG* – Part IV.

					BRKGA		BRKGA-HCB		BRKGA-IG*		RIG*		Running
Instance	$\|V\|$	$\|E\|$	Dens. (%)	γ	Best	Average	Best	Average	Best	Average	Best	Average	time (seconds)
p_hat1000-3	1000	371,746	74.42	0.950	**210**	**210.00**	**210**	**210.00**	**210**	**210.00**	209	208.30	87.93
p_hat1500-2	1500	568,960	50.61	0.800	**642**	**642.00**	**642**	**642.00**	**642**	**642.00**	**642**	**642.00**	274.25
san200_0.7_1	200	13,930	70.00	0.950	**57**	**57.00**	**57**	**57.00**	**57**	**57.00**	**57**	55.80	3.91
san200_0.7_2	200	13,930	70.00	0.950	**34**	**34.00**	**34**	33.60	**34**	**34.00**	**34**	**34.00**	2.43
san200_0.9_1	200	17,910	90.00	0.990	74	70.40	**78**	**78.00**	**78**	**78.00**	**78**	77.00	6.76
san200_0.9_2	200	17,910	90.00	0.999	55	50.40	58	58.00	**60**	**60.00**	55	46.20	3.97
san200_0.9_3	200	17,910	90.00	0.999	37	36.90	42	39.40	**44**	**42.60**	37	36.30	2.80
san400_0.5_1	400	39,900	50.00	0.500	**400**	**400.00**	**400**	**400.00**	**400**	**400.00**	**400**	**400.00**	29.40
san400_0.7_1	400	55,860	70.00	0.950	**201**	**201.00**	**201**	**201.00**	**201**	**201.00**	**201**	**201.00**	23.28
san400_0.7_2	400	55,860	70.00	0.950	**62**	**62.00**	**62**	**62.00**	**62**	**62.00**	**62**	**62.00**	8.03
san400_0.7_3	400	55,860	70.00	0.950	**40**	37.20	39	36.80	**40**	**38.90**	38	36.40	6.90
san1000	1000	250,500	50.15	0.800	**562**	**562.00**	**562**	**562.00**	**562**	**562.00**	**562**	**562.00**	126.67
sanr200_0.7	200	13,868	69.69	0.800	**73**	72.50	72	72.00	**73**	**72.90**	72	72.00	4.74
sanr200_0.9	200	17,863	89.76	0.950	91	91.00	**92**	91.50	**92**	**92.00**	91	90.90	7.80
sanr400_0.5	400	39,984	50.11	0.800	**32**	31.40	**32**	31.80	**32**	**32.00**	31	30.00	4.64
sanr400_0.7	400	55,869	70.01	0.950	30	29.30	**32**	31.20	**32**	**31.80**	30	28.70	5.01

Table 7
Comparative performance statistics for algorithms BRKGA, BRKGA-HCB, BRKGA-IG*, and RIG*.

	BRKGA	BRKGA-HCB	BRKGA-IG*	RIG*
#Best	44	52	**97**	36
#BestAvg	32	35	**96**	27
SumBest	356	404	**575**	283
AvgDevRuns (%)	3.33	3.29	**0.95**	4.99
AvgDev (%)	2.27	2.07	**0.07**	3.35
AvgDevAvg (%)	2.48	2.45	**0.06**	4.17
Score	84	87	**4**	158
ScoreM	102	119	**6**	203

BRKGA-HCB or BRKGA-IG* found solutions unmatched by RIG* for 64 test problems.

Table 7 summarizes the following statistics resulting from the experiments reported in Tables 3–6:

- *#Best* is the number of instances for which a given heuristic found the best overall solution value. The higher its value, the better is the performance of the corresponding heuristic.
- *#BestAvg* is the number of instances for which a given heuristic found the best average solution value. The higher its value, the better is the performance of the corresponding heuristic.
- *SumBest* is the number of runs in which a given heuristic found the best overall solution value. The higher its value, the better is the performance of the corresponding heuristic.
- *AvgDevRuns* is the average relative deviation between the solution value found by a given heuristic over all runs of some instance and the best solution value obtained for this instance over all heuristics. The smaller its value, the better is the performance of the corresponding heuristic.
- *AvgDev* is the average relative deviation between the best solution value obtained by a given heuristic for some instance and the best solution value obtained for this instance over all

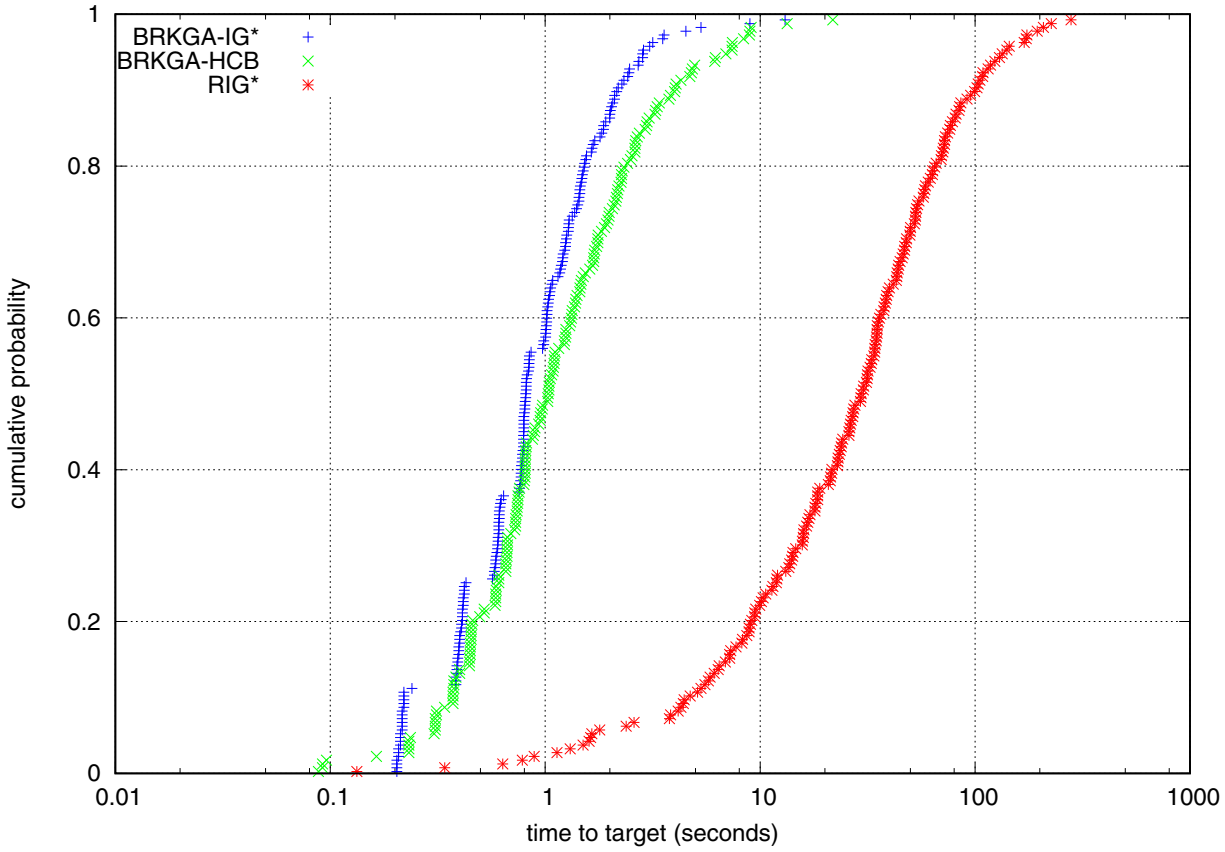


Fig. 3. Runtime distributions for the instance san200_0.9_1 with the target value set at 78 (threshold $\gamma = 0.90$).

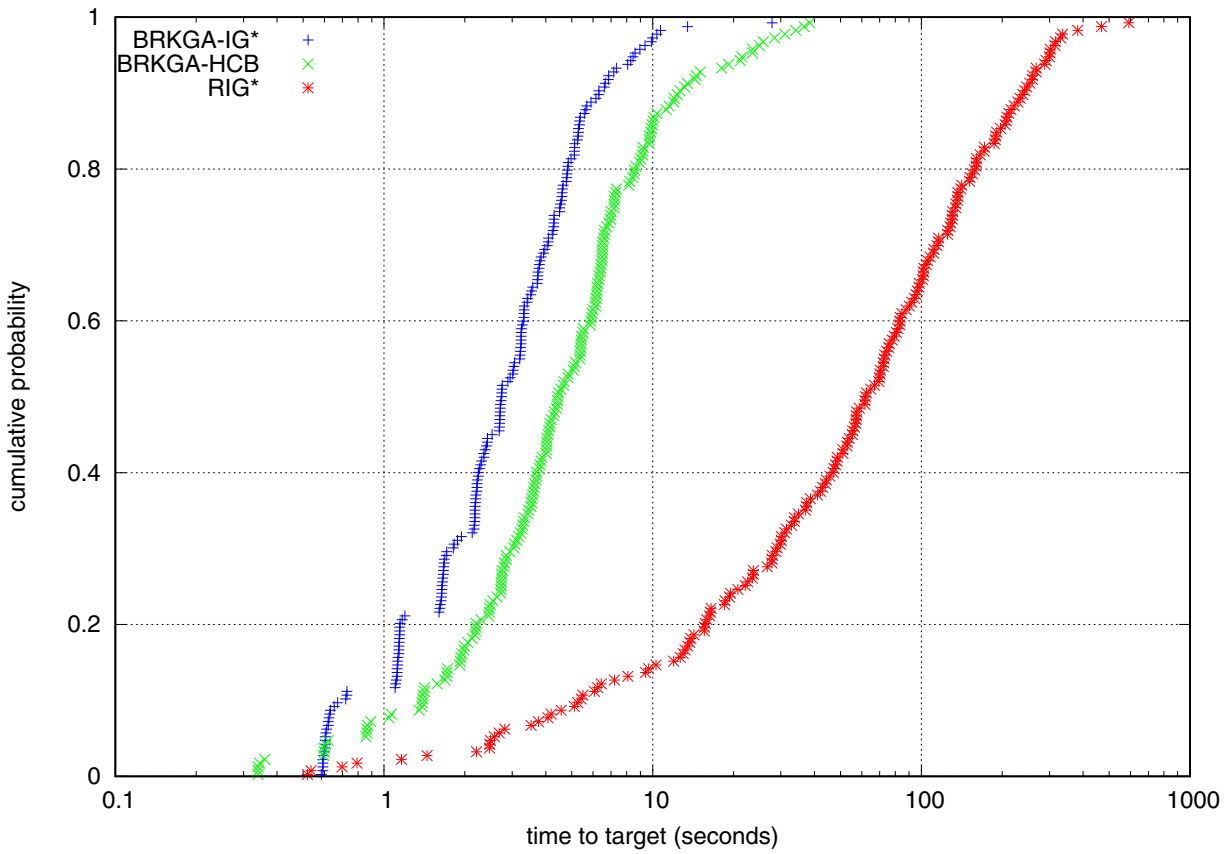


Fig. 4. Runtime distributions for the instance C500.9 with the target value set at 56 (threshold $\gamma = 0.999$).

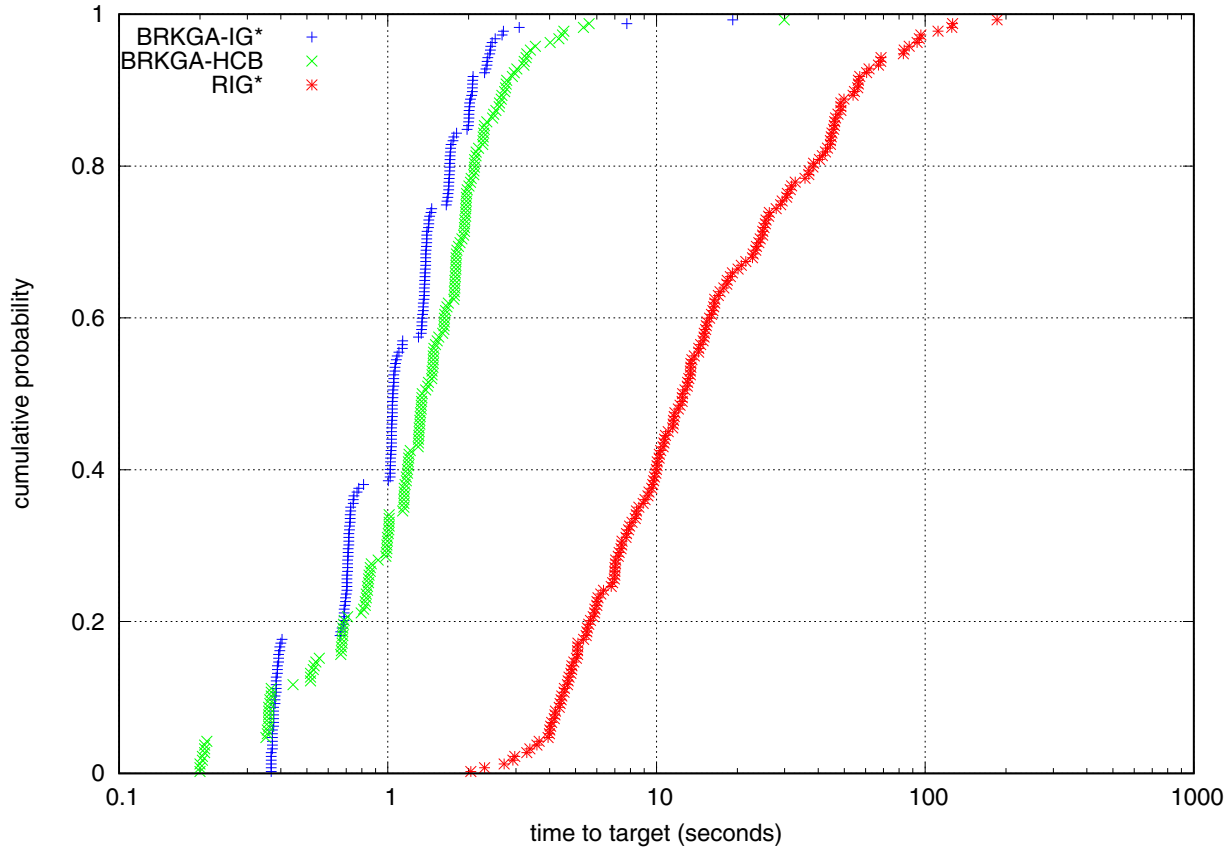


Fig. 5. Runtime distributions for the instance gen400_p0.9_65 with the target value set at 51 (threshold $\gamma = 0.999$).

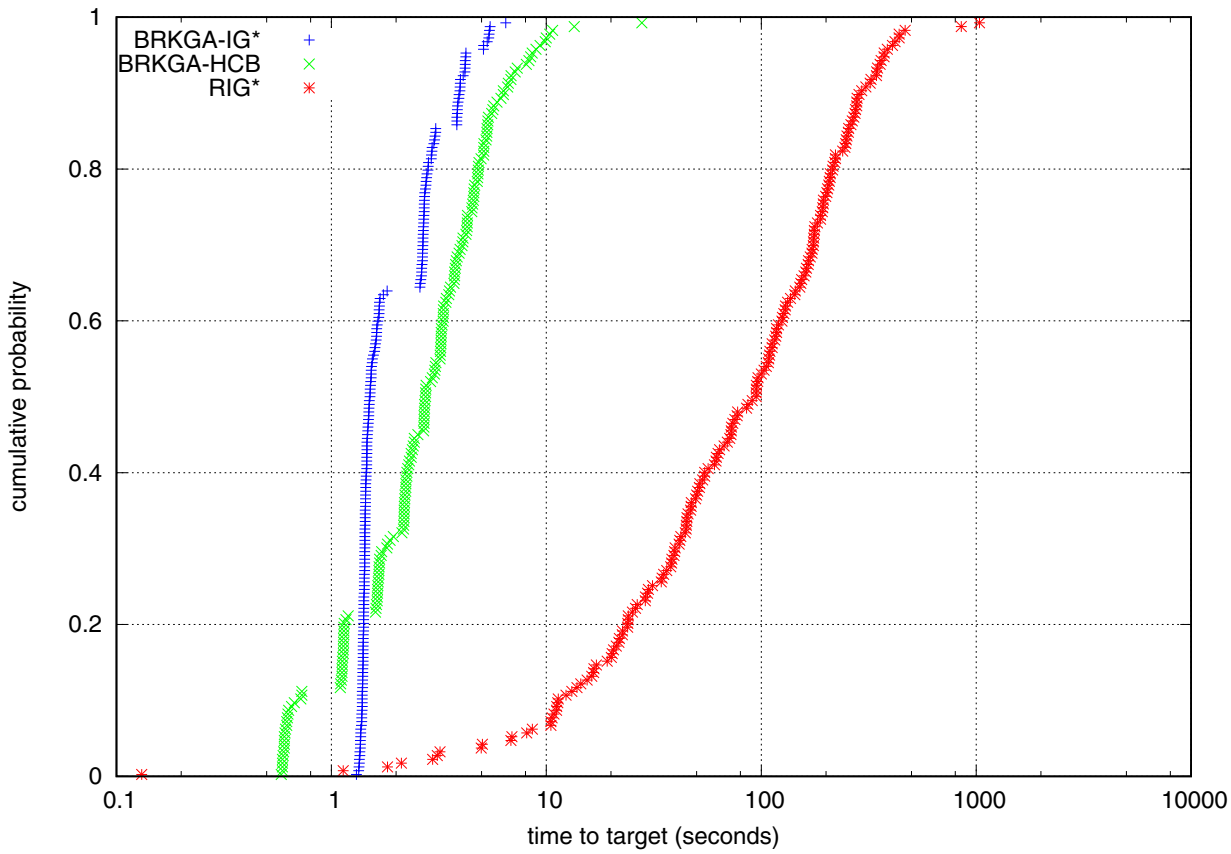


Fig. 6. Runtime distributions for the instance frb30-15-4 with the target value set at 56 (threshold $\gamma = 0.95$).

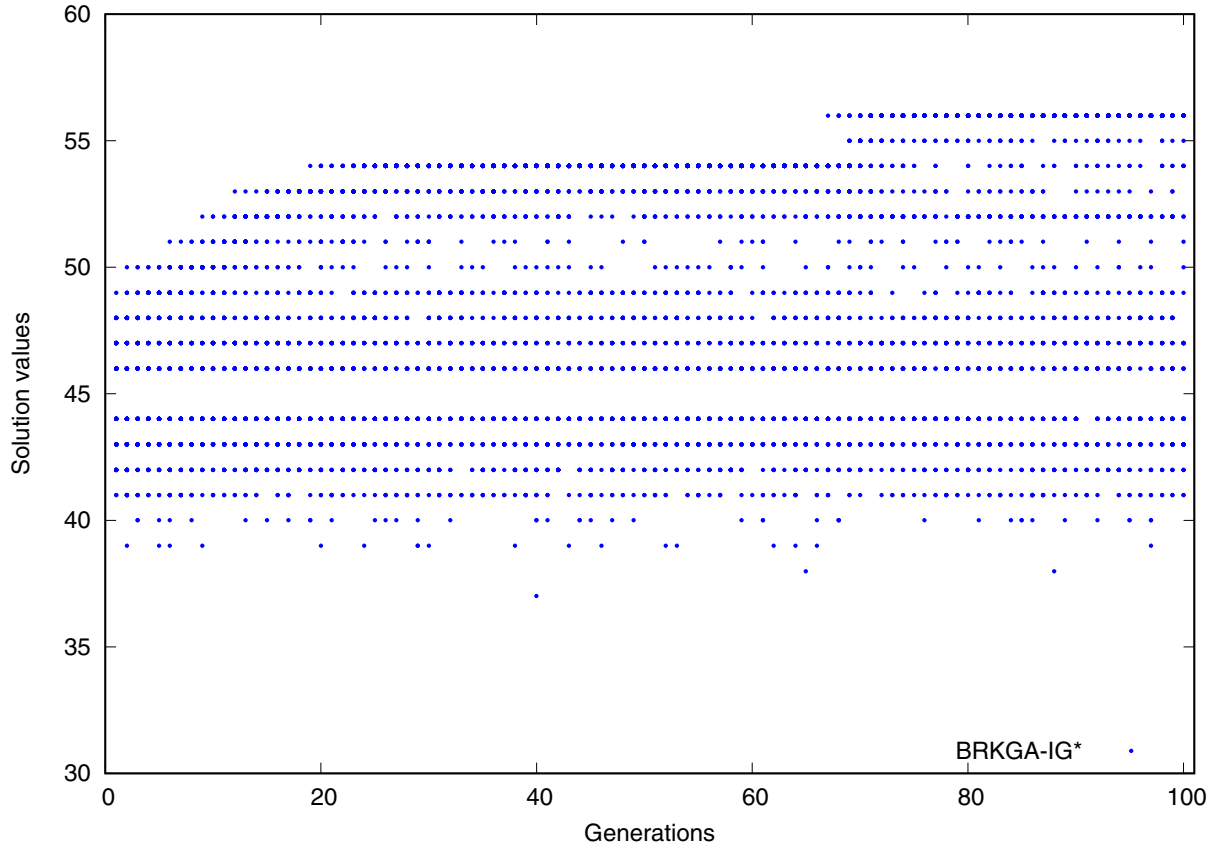


Fig. 7. Population evolution for instance gen400_p0.9_65: best value found by BRKGA-IG* after 33.3 seconds (100 generations) is 56 (threshold $\gamma = 0.999$).

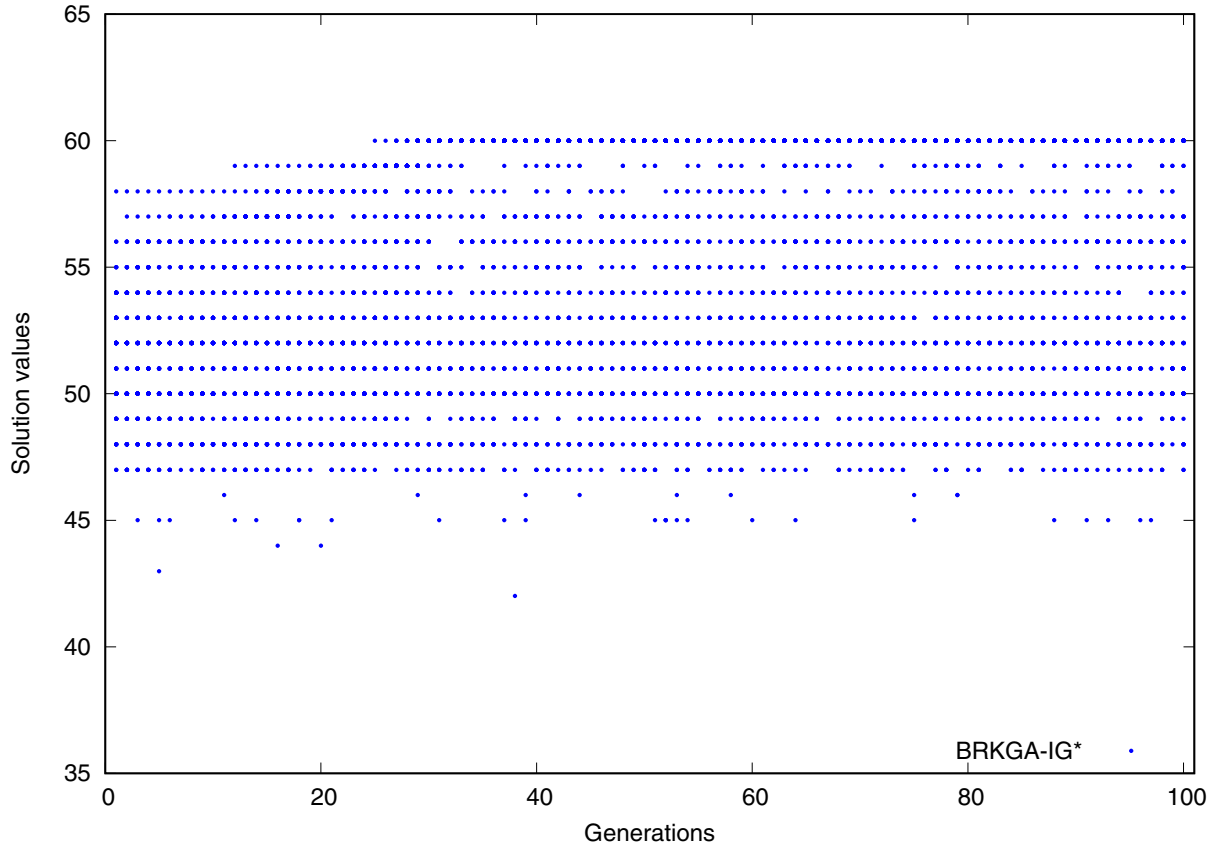


Fig. 8. Population evolution for instance frb30-15-4: best value found by BRKGA-IG* after 147.99 seconds (100 generations) is 60 (threshold $\gamma = 0.95$).

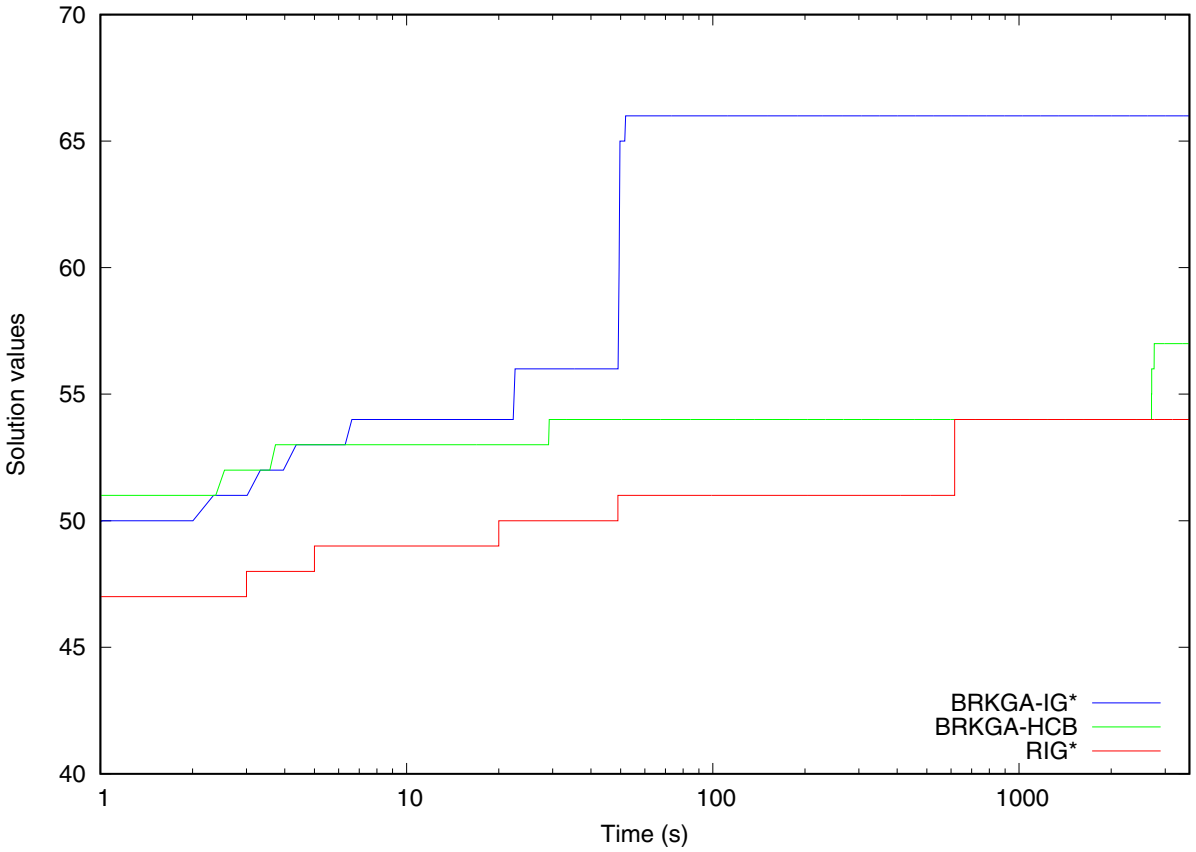


Fig. 9. Evolution of the best solution value found by BRKGA-IG* and the other algorithms along the 3600 first seconds of running time for instance gen400_p0.9_65: best solution value obtained by BRKGA-IG* is 66 (threshold $\gamma = 0.999$).

Table 8
Summary of computational results for each strategy on instance san200_0.9_1. Each run was made to stop when a solution as good as the target solution value 79 was found (threshold $\gamma = 0.90$). For each strategy, the table shows the distribution of the running times by quartile. For each quartile, the table gives the average running times in seconds over all runs in that quartile. The average running times over the 200 runs are also given for each strategy.

	Average running times in quartile (seconds)				
Strategy	1st	2nd	3rd	4th	Average
BRKGA-IG* without restarts	118.48	906.97	3341.26	–	–
BRKGA-IG* with restart(100)	**39.51**	**133.93**	**330.57**	**799.44**	**325.86**
BRKGA-IG* with restart(200)	81.07	232.08	438.30	1038.03	447.37
BRKGA-IG* with restart(500)	95.23	305.23	621.06	1452.35	618.47

Table 9
Test instances used in the tuning experiment with IRACE for sparse graphs.

Instance	$\|V\|$	$\|E\|$	Threshold γ
CA-GrQc	5242	14,496	0.4
CA-GrQc	5242	14,496	0.6
CA-GrQc	5242	14,496	0.8
Harvard500	500	2636	0.4
Harvard500	500	2636	0.6
Harvard500	500	2636	0.8
USAir97	332	2126	0.4
USAir97	332	2126	0.6
USAir97	332	2126	0.8

heuristics. The smaller its value, the better is the performance of the corresponding heuristic.

- *AvgDevAvg* is the average relative deviation between the average solution value obtained by a given heuristic for some instance and the best average solution value obtained for this instance over all heuristics. The smaller its value, the better is the performance of the corresponding heuristic.
- *Score* is the sum over all instances of the number of approaches that provided a solution strictly better than that obtained by a given heuristic. The smaller the value of *Score*, the better is the performance of the corresponding heuristic.
- *ScoreAvg* is the sum over all instances of the number of approaches that found an average solution value strictly better than that obtained by a given heuristic for some instance. The smaller its value, the better is the performance of the corresponding heuristic.

The results in Table 7 show that variant BRKGA-IG* consistently obtains the best performance statistics over all criteria considered. BRKGA-IG* found 97 best solution values and 96 best average solution values out of the 100 instances. BRKGA-IG* also obtained

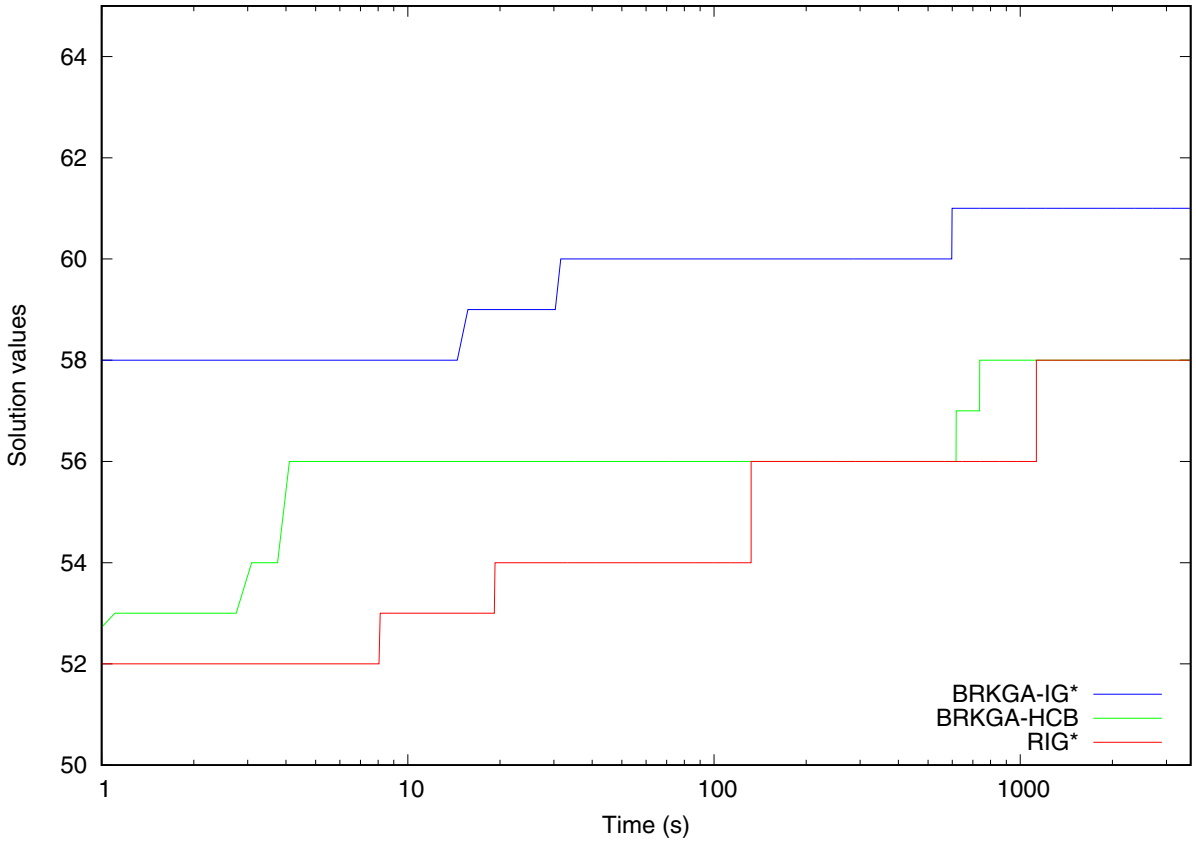


Fig. 10. Evolution of the best value found by BRKGA-IG* and the other algorithms along the 3600 first seconds of running time for instance frb30-15-4: best solution value obtained by BRKGA-IG* is 61 (threshold $\gamma = 0.95$).

Table 10
Comparative running times in seconds for algorithms AlgF3 (original times multiplied by 0.65) and AlgF4 (original times multiplied by 0.65), and BRKGA-IG* with restart(100).

Instance	$\|V\|$	$\|E\|$	dens. (%)	γ	$\omega_\gamma(G)$	Running times (seconds) AlgF3	AlgF4	BRKGA-IG* restart(100)
Harvard500	500	2043	1.64	0.9	23	9.11	10.08	0.07
Harvard500	500	2043	1.64	0.5	37	36.62	11.57	1.21
CA-GrQc	5242	14,496	0.10	0.9	49	200.85	207.44	2.34
CA-GrQc	5242	14,496	0.10	0.5	81	328.24	406.12	–
USAir97	332	2126	3.87	0.9	35	13.26	4.42	0.22
USAir97	332	2126	3.87	0.5	67	10.40	1.88	0.53
Email	1133	5451	0.85	0.9	13	–	345.41	0.17
Email	1133	5451	0.85	0.5	25	809.25	–	1.48
SmallW	396	994	1.27	0.9	11	4.74	1.88	0.06
SmallW	396	994	1.27	0.5	28	3.71	1.76	0.28
Erdos971	429	1312	1.43	0.9	8	13.00	2.47	0.05
Erdos971	429	1312	1.43	0.5	23	3.97	478.59	0.23

the solutions that led to the smallest values for all statistics reporting average relative deviations from the best solution values.

In the next experiment, we evaluate and compare the run time distributions (or time-to-target plots – or ttt-plots, for short) of algorithms BRKGA-HCB, BRKGA-IG*, and RIG*. 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 (1998) as a way to characterize the running times of stochastic local search algorithms for combinatorial optimization. In this experiment, the three algorithms were made to stop whenever a solution with cost smaller than or equal to a given target value was found. The heuristics were run 200 times each, with different initial seeds for the pseudo-random number generator. Next, the empirical probability distributions of the time taken by each heuristic to find the target value are plotted. To plot the empirical distribution for each heuristic, we followed the methodology proposed by Aiex, Resende, and Ribeiro (2002, 2007). 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.

Time-to-target plots for instance san200_0.9_1 are shown in Fig. 3, with the target set to 78, which corresponds to the best value in Table 6. Time-to-target plots for instance C500.9 are shown in Fig. 4, with the target set to 56, which corresponds to the average solution value obtained by heuristic BRKGA-HCB in Table 3. Time-to-target plots for instance gen400_p0.9_65 are

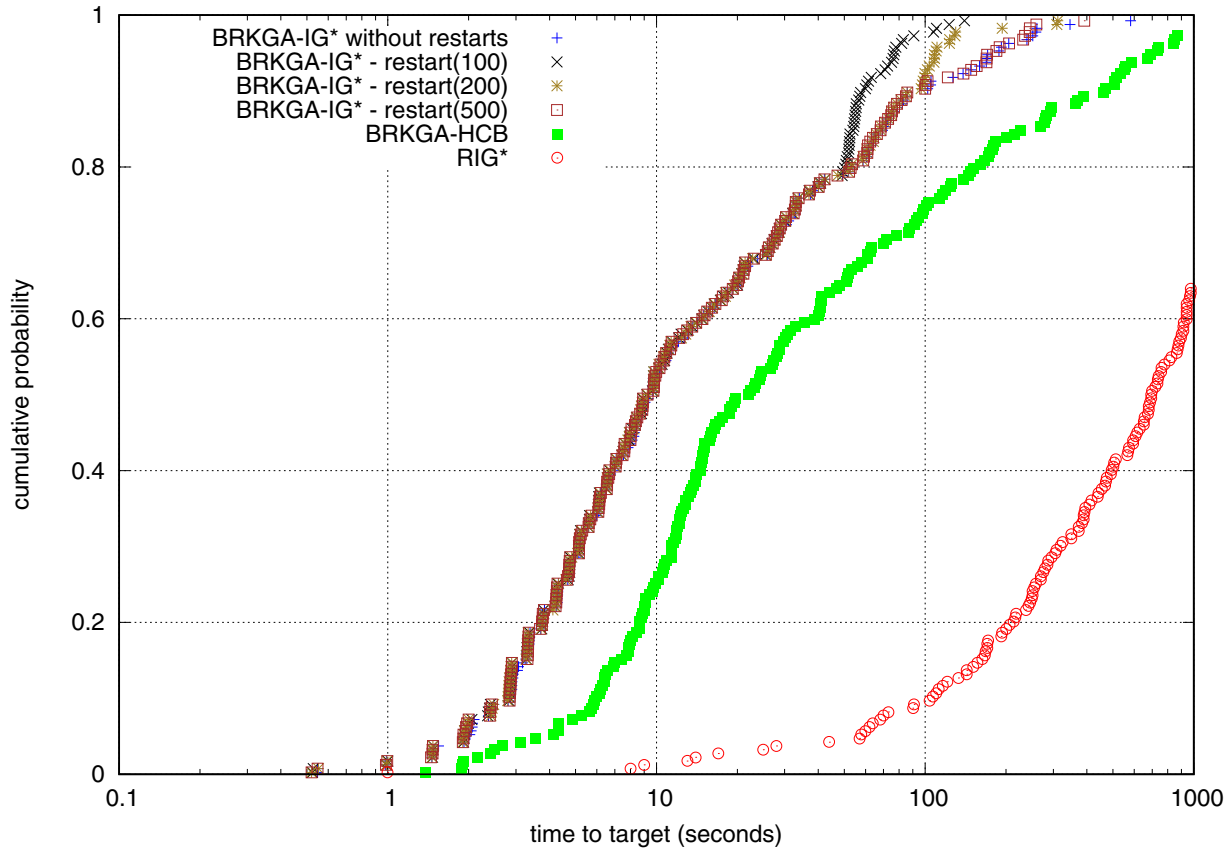


Fig. 11. Runtime distributions for heuristic BRKGA-IG* with restart(κ) strategies on instance C500.9 with the running time limited to 1000 seconds and target value set at 57 (threshold $\gamma = 0.999$).

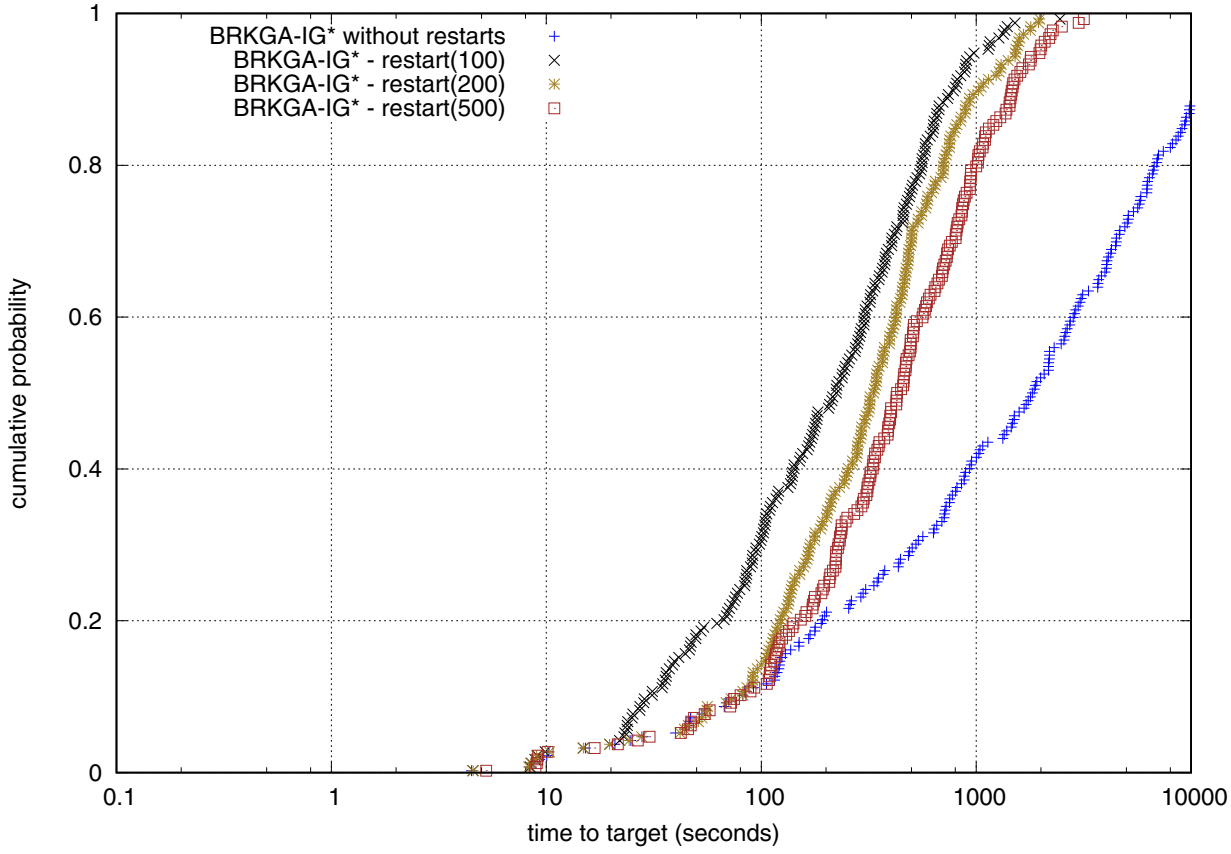


Fig. 12. Runtime distributions for heuristic BRKGA-IG* with restart(κ) strategies on instance san200_0.9_1 with the running time limited to 10,000 seconds and target value set at 79 (threshold $\gamma = 0.90$).

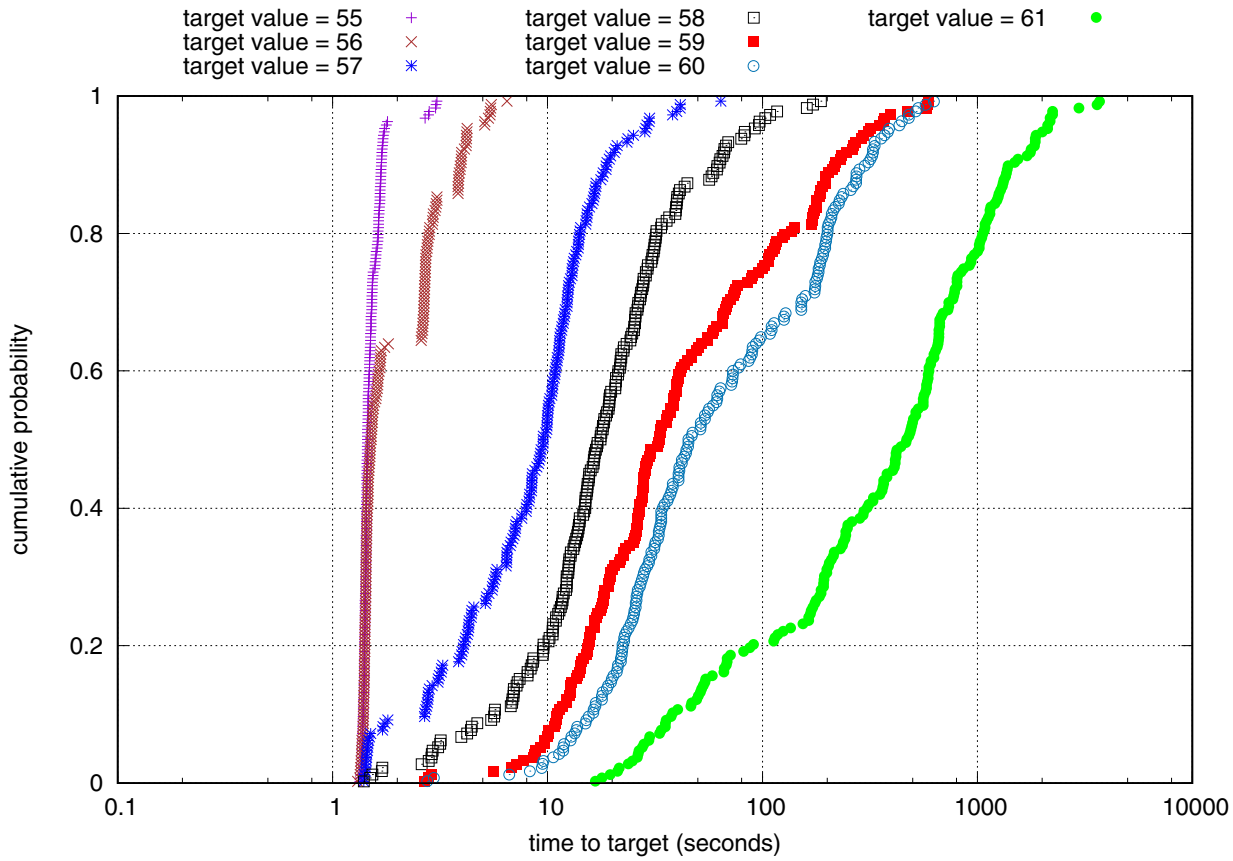


Fig. 13. Runtime distributions for heuristic BRKGA-IG* with restart(100) strategy on instance frb30-15-4 with the running time limited to 10,000 seconds as the target value set increases from 55 to 61 (threshold $\gamma = 0.95$).

shown in Fig. 5, with the target set to 51, which corresponds to the best value obtained by heuristic RIG* in Table 5. Also, time-to-target plots for instance frb30-15-4 are shown in Fig. 6, with the target set to 56, which corresponds to the average solution value obtained by heuristic BRKGA-HCB in Table 4. These plots show that heuristic BRKGA-IG* is able to find with higher probability solutions as good as the target in smaller running times.

Figs. 7 and 8 illustrate the evolution of the solution population along 100 generations of BRKGA-IG* for one execution of instances gen400_p0.9_65 and frb30-15-4, respectively. They show that the biased random-key genetic algorithm is able to continuously evolve the solution population and to improve the best solution value.

Figs. 9 and 10 illustrate how the best solutions found by the three algorithms evolve along the first 3600 seconds of processing time, for the same two instances gen400_p0.9_65 and frb30-15-4, respectively. They show that BRKGA-IG* systematically finds better solutions faster than the other algorithms. The best solution obtained by BRKGA-IG* is better than that found by RIG* and BRKGA-HCB most of the time along the runs displayed in these figures.

Restart strategies are able to reduce the running times to reach target solution values. We applied to heuristic BRKGA-IG* the same type of restart(κ) strategy discussed in Resende and Ribeiro (2011, 2016) (see also Interian & Ribeiro (2017)), in which the population is entirely renewed after κ generations have been performed without improvement in the best solution found. We evaluated the performance of restart(κ) strategies for $\kappa = 100, 200, 500$. Time-to-target plots for instance C500.9 are displayed in Fig. 11 with 200 runs of each strategy restart(κ). Each run was limited to 1000 seconds and the target value was set to 57. Fig. 11 shows that RIG* failed to reach the target within the time limit in 71 runs and BRKGA-HCB failed to reach the target within the time limit in 4 runs. For this instance, strategy restart(100) presented the best results, i.e., the leftmost plots.

The next experiment addresses the behavior of heuristic BRKGA-IG* with harder target values. Fig. 12 displays time-to-target plots for all variants of BRKGA-IG*, with and without restarts, on instance san200_0.9_1 with the target value set at 79. In this experiment, each algorithm variant was run 200 times, with the running time limited to 10,000 seconds. Again, BRKGA-IG* with restart(100) presented the best behavior.

Resende and Ribeiro (2011, 2016) observed that the effect of restart strategies can be mainly noticed in the longest runs. As an example, Table 8 illustrates the results obtained by the restart strategies on instance san200_0.9_1, considering 200 runs of each algorithm variant with the target value set to 79. We consider the column corresponding to the fourth quartile in this table, whose entries correspond to those in the heavy tails of the runtime distributions. The restart strategies affect all quartiles of the distributions, which is a desirable result. Compared to the strategy without restarts, the restart(100) strategy was able to reduce not only the average running time in the fourth quartile, but also in the other quartiles. The best results for each quartile are highlighted in boldface. Strategy BRKGA-IG* with restart(100) clearly outperformed all other variants tested, with the smallest average running times. We notice that BRKGA-IG* without restarts failed to reach the target within the time limit in 23 runs.

Fig. 13 displays time-to-target plots for BRKGA-IG* with restart (100) strategy on instance frb30-15-4 with the running time limited to 10000 seconds as the target value increases from 55 to 61. Fig. 14 shows the average running time (in seconds) over 200

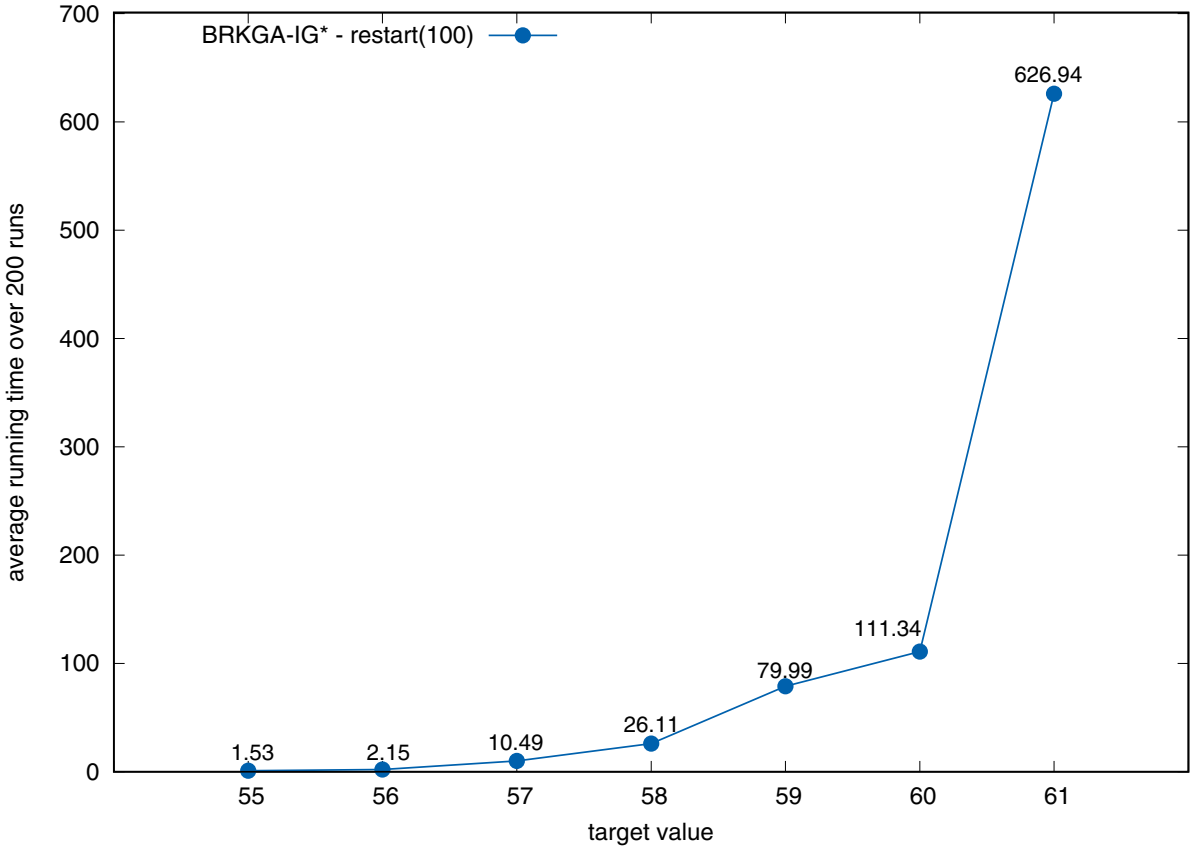


Fig. 14. Average running times (seconds) for heuristic BRKGA-IG* with restart(100) strategy on instance frb30-15-4 with the running time limited to 10,000 seconds as the target value set ranges from 55 to 61 (threshold $\gamma = 0.95$).

runs for each target. As expected, the running time grows fast as the target increases towards the optimal value.

The experiments reported in this section showed that the biased random-key genetic algorithm BRKGA-IG* with the restart(100) strategy obtained the best results among all tested variants.

6.3. Experiments on sparse graphs

In Section 6.2, we concluded that variant BRKGA-IG* with strategy restart(100) presented the best numerical results on dense graphs. In this section, we compare this approach with algorithms AlgF3 and AlgF4, originally proposed in Veremyev et al. (2016). Since the original source codes of AlgF3 and AlgF4 were not available and the experiments reported in Veremyev et al. (2016) have been made on a different processor, we should consider an approximate scale ratio to compare the speed of the two processors. Regarding the single-rating performance, since the algorithms were tested on single-thread environments, the corresponding ratio for the two CPU models is $1067/1646 \approx 0.65$, according to the PassMark benchmark (PassMark, 2018).

The performance of algorithms to the maximum quasi-clique problem is very sensitive to the density of the input graph. Since heuristic BRKGA-IG* did not performed well for sparse graphs with the parameter $\alpha = 0.01$ set as determined in the experiment reported in Section 6.1, we performed a new tuning experiment for this parameter for the case of sparse graphs. Table 9 displays the characteristics of the nine instances of the University of Florida Sparse Matrix Collection (Davis & Hu, 2011) used for tuning parameter α in the range $0.01, 0.02, \ldots, 0.20$. The heuristic was run 1000 times, with all other parameters fixed as before. The experiment determined $\alpha = 0.09$ as the most appropriate value for this parameter.

Algorithms AlgF3, AlgF4, and BRKGA-IG* with restart(100) were compared on six sparse instances with two values of the threshold γ for each of them. Each algorithm was made to stop when a solution with cardinality given by the bound $\omega_\gamma(G)$ was reached (Veremyev et al., 2016). The running times for each algorithm are reported in Table 10. The running times for algorithms AlgF3 and AlgF4 are those reported by Veremyev et al. (2016), multiplied by the scale factor 0.65. The last column gives the average running time over ten runs of BRKGA-IG* with restart(100) for $\alpha = 0.09$, limited to one hour for each run. The running times of algorithms AlgF3 and AlgF4 were also limited to one hour for each instance. Blank cells correspond to cases where none of the ten runs reached a solution of cardinality $\omega_\gamma(G)$. For all other cells, all ten runs found a γ-clique with $\omega_\gamma(G)$ vertices. These results show that the biased random-key genetic algorithm BRKGA-IG* with the restart(100) strategy proposed in this work was faster than AlgF3 and AlgF4 for all but one of the test instances (CA-GrQc with the threshold $\gamma = 0.5$), for which it was not able to find a solution as good as the target in less than one hour of computation in any of the ten runs. This was most likely due to the very small density of this instance, for which AlgF3 and AlgF4 also took very long.

7. Concluding remarks

Given a graph $G = (V, E)$ and a threshold $\gamma \in (0, 1]$, the maximum cardinality quasi-clique problem consists in finding a maximum 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.

In this work, we proposed a biased random-key genetic algorithm for finding approximate solutions to the maximum quasi-clique problem, using two different decoders. The decoder based on an optimized iterated greedy constructive heuristic led to the best numerical results. We also showed that the use of a restart strategy significantly contributed to improve the robustness and the efficiency of the algorithm. The resulting BRKGA-IG* heuristic with restart(100) strategy outperformed different variants of the algorithm, as well as the restarted optimized iterated greedy (RIG*) construction/destruction heuristic that originally reported the best results in the literature for dense graphs.

In addition, the newly proposed BRKGA-IG* with restart(100) approach was also compared with the exact algorithms AlgF3 and AlgF4 of Veremyev et al. (2016) used as a heuristics with time limits on their running times. Also in this case, BRKGA-IG* with restart(100) applied to sparse graphs outperformed both mixed integer programming approaches, finding target solution values in much smaller running times.

All the input data for the test instances used in the experiments reported in this work are available in Mendeley (see Pinto et al. (2017)), together with the resulting detailed numerical results.

Acknowledgment

Work of Celso C. Ribeiro was partially supported by CNPq research grant 303958/2015-4 and by FAPERJ research grant E-26/201.198/2014. Work of Alexandre Plastino was partially supported by CNPq research grant 308369/2015-7.

References

Abello, J., Pardalos, P. M., & Resende, M. G. C. (1999). On maximum clique problems in very large graphs. In J. M. Abello, & J. S. Vitter (Eds.), *External memory algorithms* (pp. 119–130). American Mathematical Society.

Abello, J., Resende, M., & Sudarsky, S. (2002). Massive quasi-clique detection. In S. Rajsbaum (Ed.), *Proceedings of the fifth Latin American symposium on theoretical informatics: Latin 2002*. In *Lecture Notes in Computer Science: Vol. 2286* (pp. 598–612). Berlin: Springer.

Aiex, R., Resende, M., & Ribeiro, C. C. (2002). Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics, 8*, 343–373.

Aiex, R., Resende, M., & Ribeiro, C. C. (2007). TTTPLOTS: A Perl program to create time-to-target plots. *Optimization Letters, 1*, 355–366.

Alfaro-Fernández, P., Ruiz, R., Pagnozzi, F., & Stützle, T. (2017). Exploring automatic algorithm design for the hybrid flowshop problem. In *Proceedings of the twelfth metaheuristics international conference* (pp. 201–203). Barcelona.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing, 6*, 154–160.

BHOSLIB (2014). Benchmarks with hidden optimum solutions for graph problems. http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm Last accessed on November 14, 2017.

Bouamama, S., & Blum, C. (2017). A population-based iterated greedy algorithm for the knapsack problem with setup. In *Proceedings of the twelfth metaheuristics international conference* (pp. 558–565). Barcelona.

Brandão, J. S., Noronha, T. F., Resende, M. G. C., & Ribeiro, C. C. (2015). A biased random-key genetic algorithm for single-round divisible load scheduling. *International Transactions Operational Research, 22*, 823–839.

Brandão, J. S., Noronha, T. F., Resende, M. G. C., & Ribeiro, C. C. (2017). A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems. *International Transactions Operational Research, 27*, 1061–1077.

Brunato, M., Hoos, H., & Battiti, R. (2008). On effectively finding maximal quasi–cliques in graphs. In V. Maniezzo, R. Battiti, & J.-P. Watson (Eds.), *Proceedings of the second international conference learning and intelligent optimization, LION 2007*. In *Lecture Notes in Computer Science: Vol. 5313* (pp. 41–55). Berlin: Springer.

Chaves, A. A., Lorena, L., Senne, E., & Resende, M. G. C. (2016). Hybrid method with CS and BRKGA applied to the minimization of tool switches problem. *Computers & Operations Research, 67*, 174–183.

Davis, T. A., & Hu, Y. (2011). The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software, 38*, 1:1–1:25.

DIMACS (2016). DIMACS Implementation Challenges. http://dimacs.rutgers.edu/Challenges/ Last access on November 14, 2017.

FICO (2017). FICO Xpress Optimization Suite 7.6. http://www.fico.com/en/products/fico-xpress-optimization-suite Last access on November 14, 2017.

Gonçalves, J. F., & Resende, M. G. C. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics, 17*, 487–525.

Gonçalves, J. F., Resende, M. G. C., & Toso, R. F. (2014). An experimental comparison of biased and unbiased random-key genetic algorithms. *Pesquisa Operacional, 34*, 143–164.

Hoos, H., & Stützle, T. (1998). Evaluation of Las Vegas algorithms – Pitfalls and remedies. In G. Cooper, & S. Moral (Eds.), *Proceedings of the fourteenth conference on uncertainty in artificial intelligence* (pp. 238–245). Madison.

Interian, R., & Ribeiro, C. (2017). A GRASP heuristic using path-relinking and restarts for the Steiner traveling salesman problem. *International Transactions in Operational Research, 24*, 1307–1323.

(1996). Cliques, coloring, and satisfiability: Second DIMACS implementation challenge. In D. J. Johnson, & M. A. Trick (Eds.), *Dimacs series in discrete mathematics and theoretical computer science: Vol. 26*. Boston: American Mathematical Society.

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller, & J. W. Thatcher (Eds.), *Complexity of computer computations* (pp. 85–103). Boston: Springer.

López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., & Birattari, M. (2011). The IRACE package: Iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium.

Maschler, J., Hackl, T., Riedler, M., & Raidl, G. R. (2017). Enhanced iterated greedy metaheuristic for the particle therapy patient scheduling problem. In *Proceedings of the twelfth metaheuristics international conference* (pp. 118–127). Barcelona.

Noronha, T. F., Resende, M. G. C., & Ribeiro, C. C. (2011). A biased random-key genetic algorithm for routing and wavelength assignment. *Journal of Global Optimization, 50*, 503–518.

Oliveira, A. B., Plastino, A., & Ribeiro, C. C. (2013). Construction heuristics for the maximum cardinality quasi-clique problem. In *Abstracts of the tenth metaheuristics international conference (mic 2013)* (p. 84). Singapore.

PassMark (2018). CPU Benchmarks. https://www.cpubenchmark.net/compare.php Last access on March 14, 2018.

Pattillo, J., Veremyev, A., Butenko, S., & Boginski, V. (2013). On the maximum quasi–clique problem. *Discrete Applied Mathematics, 161*, 244–257.

Pérez Cáceres, L., López-Ibáñez, M., & Stützle, T. (2014). An analysis of parameters of IRACE. In *Proceedings of the fourteenth European conference on evolutionary computation in combinatorial optimization*. In *Lecture Notes in Computer Science: Vol. 8600* (pp. 37–48). Berlin: Springer.

Pinto, B. Q., Plastino, A., Ribeiro, C. C., & Rosseti, I. (2015). A biased random-key genetic algorithm to the maximum cardinality quasi-clique problem. In *Abstracts of the eleventh metaheuristics international conference*. Agadir.

Pinto, B. Q., Ribeiro, C. C., Rosseti, I., & Plastino, A. (2017). Input data and detailed numerical results for 'A biased random-key genetic algorithm for the maximum quasi-clique problem'. https://data.mendeley.com/datasets/khdncrwb5s/1 Last accessed on November 14, 2017. 10.17632/khdncrwb5s.1.

Pullan, W., Mascia, F., & Brunato, M. (2011). Cooperating local search for the maximum clique problem. *Journal of Heuristics, 17*, 181–199.

Resende, M. G. C., & Ribeiro, C. C. (2011). Restart strategies for GRASP with path-relinking heuristics. *Optimization Letters, 5*, 467–478.

Resende, M. G. C., & Ribeiro, C. C. (2016). *Optimization by GRASP*. Boston: Springer.

Ribeiro, A. G., Oliveira, B. B., Carravilla, M. A., & Oliveira, J. F. (2017). A biased random-key genetic algorithm for the car rental vehicle-reservation assignment problem. In *Proceedings of the twelfth metaheuristics international conference* (pp. 301–310). Barcelona.

Ribeiro, C. C., & Riveaux, J. A. (2018). An exact algorithm for the maximum quasi-clique problem. Submitted for publication.

Ruiz, E., Albareda-Sambola, M., Fernández, E., & Resende, M. G. (2015). A biased random-key genetic algorithm for the capacitated minimum spanning tree problem. *Computers & Operations Research, 57*, 95–108.

Ruiz, R., & Stützle, T. (2006). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*, 2033–2049.

Spears, W., & de Jong, K. (1991). On the virtues of parameterized uniform crossover. In R. Belew, & L. Booker (Eds.), *Proceedings of the fourth international conference on genetic algorithms* (pp. 230–236). San Mateo: Morgan Kaufman.

Toso, R. F., & Resende, M. G. C. (2015). A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software, 30*, 81–93.

Veremyev, A., Prokopyev, O. A., Butenko, S., & Pasiliao, E. L. (2016). Exact MIP-based approaches for finding maximum quasi-cliques and dense subgraphs. *Computational Optimization and Applications, 64*, 177–214.

ANEXO B – Pinto, B. Q., Ribeiro, C. C., Riveaux, J. A., Rosseti, I. "Improving a biased random-key genetic algorithm for the maximum quasi-clique problem with an exact local search". Submetido para International Transactions in Operational Research.

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 25 (2018) 1–20
DOI: xx.xxxx/itor.xxxxx

Improving a biased random-key genetic algorithm for the maximum quasi-clique problem with an exact local search

Bruno Q. Pinto[a], Celso C. Ribeiro[a,◇], Jose A. Riveaux[a,*], and Isabel Rosseti[a]

[a]*Institute of Computing, Universidade Federal Fluminense, Niterói, RJ 24210-346, Brazil.*
E-mail: jangel.riveaux@ic.uff.br [Jose]; celso@ic.uff.br [Celso]; bruno.queiroz@iftm.edu.br [Bruno]; rosseti@ic.uff.br [Isabel]

Received 30 June 2017; received in revised form XXXX; accepted XXXX

Abstract

Given a graph $G = (V, E)$ and a threshold $\gamma \in (0, 1]$, the maximum cardinality 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. This problem is NP-hard, since it admits the maximum clique problem as a special case. It has a number of applications in data mining, e.g. in social networks or phone call graphs. In this work, we propose a hybrid biased random-key genetic algorithm (BRKGA) for solving the maximum cardinality quasi-clique problem. The hybrid approach makes use of the QClique algorithm of Ribeiro and Riveaux (2017) to exactly solve the local search procedure in the reconstruction phase of the decoder. The newly proposed approach is compared with algorithm BRKGA-IG* of Pinto et al. (2017), the best heuristic in the literature at the time of writing. Computational results show that the hybrid BRKGA outperforms BRKGA-IG*.

Keywords: maximum cardinality quasi-clique problem; maximum clique problem; maximum clique problem; biased random-key genetic algorithm; metaheuristics; 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. 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' (i.e., $E(V')$ is the edge set of the graph induced in G by V').

*Author to whom all correspondence should be addressed (e-mail: jangel.riveaux@ic.uff.br).
◇Work of Celso C. Ribeiro was partially supported by CNPq research grant 303958/2015-4 and by FAPERJ research grant E-26/201.198/2014.

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.

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).

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 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 is also NP-hard, since it admits the maximum clique problem as a special case in which $\gamma = 1$, see (Pattillo et al., 2013). 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 (Brunato et al., 2008) and the analysis of massive telecommunication data sets obtained from social networks or phone call graphs (Abello et al., 2002), as well as various data mining and graph mining applications.

A few heuristics for MQCP exist in the literature, based on well known approaches such as greedy randomized algorithms and their iterated extensions (Oliveira, 2013; Oliveira et al., 2013), stochastic local search (Brunato et al., 2008), and GRASP (Abello et al., 2002). Pinto et al. (2017) proposed a biased random-key genetic algorithm for finding approximate solutions to the maximum cardinality quasi-clique problem, using two different decoders. They showed that the decoder based on an optimized iterated greedy constructive heuristic led to the best numerical results. They also showed that the use of a restart strategy significantly contributed to improve the robustness and the efficiency of the algorithm. The resulting BRKGA-IG* heuristic with restart(100) strategy achieved the best performance and outperformed the restarted optimized iterated greedy (RIG*) construction/destruction heuristic that originally reported the best results in the literature for dense graphs. BRKGA-IG* with restart(100) approach was also compared with the exact algorithms AlgF3 and AlgF4 of Veremyev et al. (2016) used as a heuristics with time limits on their running times. BRKGA-IG* with restart(100) applied to sparse graphs also outperformed these mixed integer programming approaches, finding target solution values in much smaller running times.

Ribeiro and Riveaux (2017) proposed an exact enumeration algorithm to solve the maximum quasi-clique 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 (Pattillo et al., 2013; Veremyev et al., 2016) solved by CPLEX and with the branch-and-bound algorithm proposed by Pajouh et al. (2014), in terms of both solution quality and running time.

In this work, we show that the exact enumeration algorithm QClique proposed by Ribeiro and Riveaux (2017) can be hybridized with the biased random-key genetic algorithm BRKGA-IG* developed by (Pinto et al., 2017) as a local search algorithm 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 used in the implementation of the biased random-key genetic algorithm for the maximum quasi-clique problem. The new decoder DECODER-ExactQClique based on an exact local search procedure is presented in

Section 5. Numerical results are reported in Section 6. 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 (Pattillo et al., 2013) :

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

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

$$\max \sum_{i \in V} x_i \tag{1}$$

subject to:

$$\sum_{(i,j) \in E : i<j} y_{ij} \geq \gamma \cdot \sum_{i,j \in V : i<j} y_{ij} \tag{2}$$

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

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

$$y_{ij} \geq x_i + x_j - 1, \qquad i, j = 1, \ldots, n, \quad i < j, \tag{5}$$

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

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

The objective function (1) maximizes the number of vertices in the solution. If two vertices i, 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 any edge may contribute to the density of a solution only if both of its ends are chosen to belong to this solution. Constraints (5) ensure that any existing edge $(i, j) \in E$ will contribute to the solution if both of its ends are chosen. Constraints (6) and (7) impose the binary and non-negativity requirements on the problem variables, respectively.

Veremyev et al. (2016) 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 (Pattillo et al., 2013), with all mixed integer programs solved using FICO Xpress-Optimizer (FICO, 2017) with the time limit of 3600 seconds. Ribeiro and Riveaux (2017) developed an exact algorithm based on a quasi-hereditary property and 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 approaches in the literature and that their new upper bound is consistently tighter than previously existing bounds.

3. Biased random-key genetic algorithms for maximum quasi-clique

Genetic algorithms with random keys, or random-key genetic algorithms (RKGA), were first introduced by Bean (1994) for combinatorial optimization problems whose solutions may be represented by permutation vectors. Solutions are represented as vectors of randomly generated real numbers called keys. A deterministic algorithm, called a decoder, takes as input a solution vector and associates with it a feasible solution of the combinatorial optimization problem, for which an objective value or fitness can be computed. Two parents are selected at random from the entire population to implement the crossover operation in the implementation of an RKGA. Parents are allowed to be selected for mating more than once in the same generation.

A biased random-key genetic algorithm (BRKGA) differs from an RKGA in the way parents are selected for crossover, see (Goncalves and Resende, 2011) for a review. In a BRKGA, each element is generated combining one element selected at random from the elite solutions in the current population, while the other is a non-elite solution. The selection is said to be biased because one parent is always an elite solution and has a higher probability of passing its genes to the new generation.

In the following, we summarize two variants of a biased random-key genetic algorithm for MQCP, each of them using a different decoder. Both of them evolve a population of chromosomes that consists of vectors of real numbers. Each chromosome is represented by a vector of $|V|$ components, in which each key is a real number in the range $[0, 1)$ associated with one of the vertices of the graph G. Each chromosome is decoded by an algorithm that receives the vector of keys and builds a feasible solution for MQCP, i.e., the decoder returns a γ-clique as its output. The two decoders DECODER-HCB and DECODER-IG* are described in the next section and have been originally presented in (Pinto et al., 2017).

The parametric uniform crossover scheme proposed by Spears and de Jong (1991) is used to combine two parent solutions and to produce an offspring. In this scheme, the offspring inherits each of its keys from the best fit of the two parents with a higher probability. The biased random-key genetic algorithm developed in this work does not make use of the standard mutation operator, where parts of the chromosomes are changed with small probability. Instead, the concept of mutants is used: mutant solutions are introduced in the population in each generation, randomly generated in the same way as in the initial population. 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 (Brandão et al., 2015, 2017; Noronha et al., 2011).

The $|V|$ keys in the chromosome are randomly generated in the initial population. At each generation, the population is partitioned into two sets: TOP and $REST$. The size of the population is $|TOP| + |REST|$. Subset TOP contains the best solutions in the population. Subset $REST$ is formed by two disjoint subsets: MID and BOT, with subset BOT being formed by the worst elements in the current population. As illustrated in Figure 1, the chromosomes in TOP are simply copied to the population of the next generation. The elements in BOT are replaced by newly created mutants that are placed in the new set BOT. The remaining elements of the new population are obtained by crossover, with one parent randomly chosen from TOP and the other from $REST$. This distinguishes a biased random-key genetic algorithm from the random-key genetic algorithm of Bean (1994), where both parents are selected at random from the entire population. Since a parent solution can be chosen for crossover more than once in any given generation, elite solutions have a higher probability of passing their random keys to the next

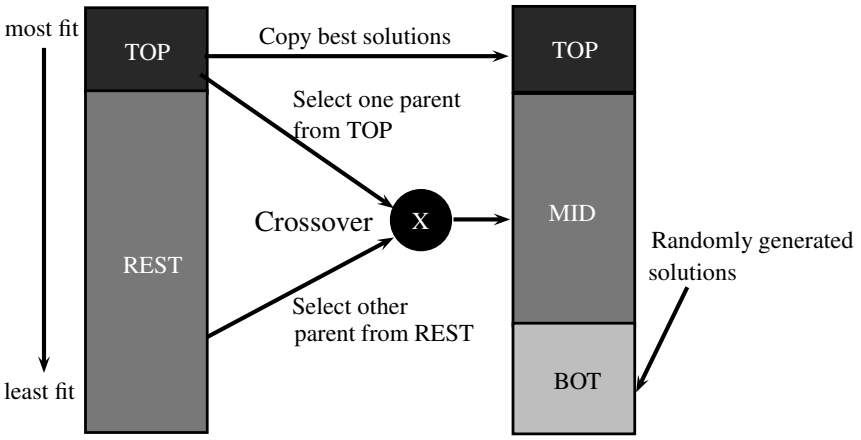


Fig. 1: Population evolution between consecutive generations of a BRKGA.

generation. In this way, $|MID| = |REST| - |BOT|$ offspring solutions are created.

The implementations of the biased random-key genetic algorithms for the maximum quasi-clique problem make use of the C++ library brkgaAPI developed by Toso and Resende (2015), which is a framework for the development of biased random-key genetic algorithms. It can also be used in parallel architectures running OpenMP.

The instantiation of the framework shown in Figure 2 to some specific optimization problem requires exclusively the development of a class implementing the decoder for this problem. This is the only problem-dependent part of the tool.

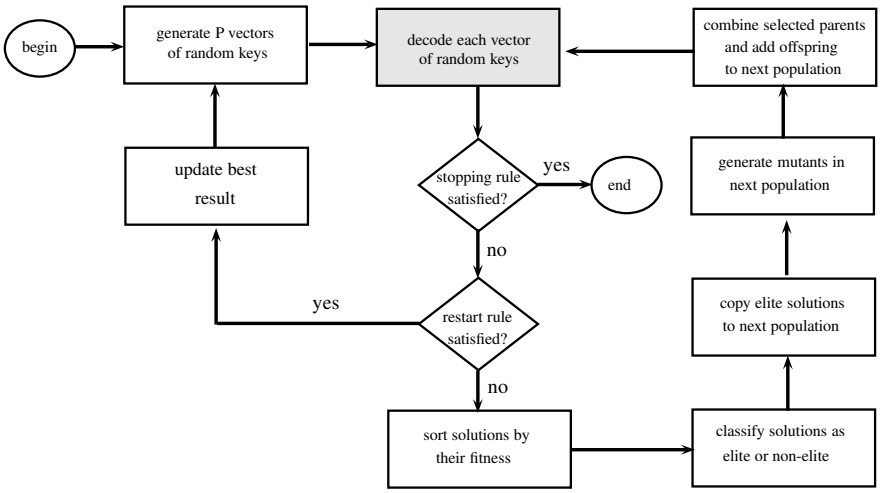


Fig. 2: BRKGA framework.

According to Goncalves et al. (2013), the BRKGA framework requires the following parameters: (a) the population size ($p = |TOP| + |REST|$); (b) the fraction pe of the population corresponding to the elite set TOP; (c) the fraction pm of the population corresponding to the mutant set BOT; (d) the probability $rhoe$ that the offspring inherits each of its keys from the best fit of the two parents; and (e) the number k of generations without improvement in the best solution until a restart is performed.

In the remainder of this work, we consider and compare two variants of a biased random-key genetic algorithm for solving MQCP, each of them based on a different decoder. Decoder DECODER-IG* was originally proposed by Pinto et al. (2017) and will be summarized in the next section. The new decoder DECODER-ExactQClique proposed in this work is based on the exact enumeration algorithm proposed by Ribeiro and Riveaux (2017) and will be presented in Section 5.

4. Decoder DECODER-IG*

We first review decoder DECODER-HCB, as presented in (Pinto et al., 2017). Each solution is associated with a set of $|V|$ random keys. The decoder receives as parameters the random keys $r_j \in [0, 1), j = 1, \ldots, |V|$. Each random key is a real number in the range $[0, 1)$ and corresponds to a vertex of the graph. Each chromosome represented by a set of random keys is decoded by an algorithm that receives the keys and builds a feasible solution to MQCP. In other words, the decoder returns a γ-clique associated with the set of random keys. Its pseudo-code is described in Algorithm 1. The roles of parameters $minsize$ and α are the same explained in (Pinto et al., 2017).

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

Decoder DECODER-IG* is an extension of the previous decoder that receives as parameters two sets of random keys $r^1_j, r^2_j \in [0, 1), j = 1, \ldots, |V|$, i.e., there are two random keys r^1_j and r^2_j associated with each vertex $j \in V$. The first set r^1 of random keys is used in the construction of the initial solution and in the reconstruction phase, while the second set r^2 is used in the destruction phase. The roles of the other parameters $minsize$, α, δ, and β are explained in (Pinto et al., 2017).

The pseudo-code of Algorithm 2 starts by creating an initial solution S' in line 1, using the decoder DECODER-HCB and the random keys $r^1_j, j \in V$. The loop in lines 2 to 10 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 2. The current solution S' is copied to S in line 3. The loop in lines 4 to 8 removes one by one the $\delta \cdot |S'|$ vertices that should be eliminated from the current solution. A restricted candidate RCL of size $\max\{minsize, \beta \cdot |CL|\}$ is created in line 5, containing the vertices with the smallest degrees in $G(S')$. The vertex with the smallest random key $r^2_j, j \in RCL$, is selected from the restricted candidate list in line 6 and eliminated from the current solution in line 7. The reconstruction phase is performed in line 9, where the current, partial solution S' is rebuilt by decoder DECODER-HCB, once again using the first set r^1 of 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. The best solution S is returned in line 11.

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

$CL \leftarrow V \setminus S$
if $S = \emptyset$ **then**
  $RCL \leftarrow \{v \in CL : |\{v' \in CL : deg_G(v') \geq deg_G(v)\}| \leq \max\{li, \alpha \cdot |CL|\}\}$
  $x \leftarrow argmin\{r_j : j \in RCL\}$
  $S \leftarrow \{x\}$
end if
while $CL \neq \emptyset$ **do**
  $CL \leftarrow \emptyset$
  for all $v \in V \setminus S$ **do**
    if $\frac{|E(S)|+deg_{G(S)}(v)}{|S|\cdot(|S|+1)/2} \geq \gamma$ **then**
      $CL \leftarrow CL \cup \{v\}$
    end if
  end for
  if $CL \neq \emptyset$ **then**
    for all $v \in CL$ **do**
      $dif_v \leftarrow deg_{G(CL)}(v) + |CL| \cdot (deg_{G(S)}(v) - \gamma \cdot (|S| + 1))$
    end for
    $RCL \leftarrow \{v \in CL : |\{v' \in CL : dif(v') \geq dif(v)\}| \leq \max\{minsize, \alpha \cdot |CL|\}\}$
    $x \leftarrow argmin\{r_j : j \in RCL\}$
    $S \leftarrow S \cup \{x\}$
  end if
end while
return S

Algorithm 2 DECODER-IG*($G, \gamma, \alpha, \delta, \beta, minsize, S, r^1, r^2$)

$S' \leftarrow$ DECODER-HCB$(G, \gamma, \alpha, minsize, \emptyset, r^1)$
repeat
  $S \leftarrow S'$
  for $k = 1$ to $\delta \cdot |S'|$ **do**
    $RCL \leftarrow \{v \in S' : |\{v' \in S' : deg_{G(S')}(v') \leq deg_{G(S')}(v)\}| \leq \max\{minsize, \beta \cdot |S'|\}\}$
    $x \leftarrow argmin\{r_j^2 : j \in RCL\}$
    $S' \leftarrow S' \setminus \{x\}$
  end for
  $S' \leftarrow$ DECODER-HCB$(G, \gamma, \alpha, minsize, S', r^1)$
until $|S'| \leq |S|$ **or** graph $G(S')$ is not connected
return S

5. New decoder based on exact local search: DECODER-ExactQClique

The main idea of the new decoder consists in replacing the reconstruction phase of decoder DECODER-IG* by an exact algorithm that optimally completes the partial solution S' obtained at the exit of the loop

in lines 4 to 8 of Algorithm 2, obtaining a maximal γ-clique S'' that contains S'.

In order to reduce the number of times the exact search algorithm is applied, an additional, $(2|V|+1)$-th random key $r^3 \in [0,1)$ will be associated to each solution. Given a parameter $\rho \in [0,1]$, local search will be applied to a solution whenever $r^3 \geq \rho$, otherwise the decoder DECODER-HCB will be used to reconstruct the solution. We note that, if $\rho = 0$ then the exact search algorithm will always be executed; if $\rho = 1$, then it will never be executed.

A slightly modified version of the QClique algorithm of Ribeiro and Riveaux (2017) will be used to complete the partial solution S' obtaining a maximal γ-clique.

Algorithm 3 describes the pseudo-code of decoder DECODER-ExactQClique. It requires the same parameters as DECODER-IG*, except for for two new parameters: the probability ρ that the exact search algorithm is applied to each partial solution and the random key r^3. The algorithm starts by creating an initial solution S' in line 1, using the decoder DECODER-HCB and the random keys $r^1_j, j \in V$. 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 $\delta \cdot |S'|$ vertices that should be eliminated from the current solution. A restricted candidate RCL of size $\max\{minsize, \beta \cdot |CL|\}$ is created in line 5, containing the vertices with the smallest degrees in $G(S')$. The vertex with the smallest random key $r^2_j, j \in RCL$, is selected in line 6 and eliminated from the current solution in line 7. The reconstruction phase starts in line 9. If the random key r^3 is greater than or equal to parameter ρ and $G(S')$ is a γ-clique, then the partial solution S' is completed in line 10 by an exact search algorithm to become a maximal γ-clique S'' that contains S'. Solution S'' is copied to S' in line 11. Solution S' is rebuilt by decoder DECODER-HCB in line 13, once again using the first set r^1 of random keys. 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 S is returned in line 15.

Algorithm 3 DECODER-ExactQClique($G, \gamma, \alpha, \delta, \beta, minsize, r^1, r^2, r^3, \rho$)

1: $S' \leftarrow$ DECODER-HCB$(G, \gamma, \alpha, minsize, \emptyset, r^1)$
2: **repeat**
3: $\quad S \leftarrow S'$
4: $\quad$ **for** $k = 1$ to $\delta \cdot |S'|$ **do**
5: $\quad\quad RCL \leftarrow \{v \in S' : |\{v' \in S' : deg_{G(S')}(v') \leq deg_{G(S')}(v)\}| \leq \max\{minsize, \beta \cdot |S'|\}\}$
6: $\quad\quad x \leftarrow argmin\{r^2_j : j \in RCL\}$
7: $\quad\quad S' \leftarrow S' \setminus \{x\}$
8: $\quad$ **end for**
9: $\quad$ **if** $r^3 \geq \rho$ **and** $dens(G(S')) \geq \gamma$ **then**
10: $\quad\quad$ Find a maximum cardinality quasi-clique S'' containing all vertices in S'.
11: $\quad\quad S' \leftarrow S''$
12: $\quad$ **end if**
13: $\quad S' \leftarrow$ DECODER-HCB$(G, \gamma, \alpha, minsize, S', r^1)$
14: **until** $|S'| \leq |S|$ **or** graph $G(S')$ is not connected
15: **return** S

We now proceed to describe how the exact algorithm QClique (Ribeiro and Riveaux, 2017) can be adapted to be used in line 10 of Algorithm 3 to optimally complete the partial solution S', generating a maximal γ-clique S''.

Algorithm 4 QClique$(G, \Pi, \gamma, LB, \Pi^*)$ (Ribeiro and Riveaux, 2017)

1: $i \leftarrow |\Pi|$
2: **if** $|E(\Pi)| + \sum_{v \in F} deg_{G(\Pi)}(v) + \min\{\frac{\sum_{v \in D} deg_G(v) - \sum_{v \in F} deg_{G(\Pi)}(v)}{2}, \binom{|V'|}{2}\} < \gamma \cdot \binom{k}{2}$ **for every** $k = i+1, \ldots, LB+1$ **then**
3: **return**
4: **end if**
5: $CL \leftarrow getCandidates(G, \Pi, \gamma)$
6: **if** $CL = \emptyset$ **then**
7: **return**
8: **end if**
9: **if** $|\Pi| + 1 > LB$ **then**
10: $LB \leftarrow |\Pi| + 1$
11: Randomly select $v \in CL$
12: $\Pi^* \leftarrow \Pi \oplus \{v\}$
13: **end if**
14: **for all** $j \in CL$ **do**
15: $\Pi' \leftarrow \Pi \oplus \{j\}$
16: QClique$(G, \Pi', \gamma, LB, \Pi^*)$
17: **end for**

The partial solution S' in line 10 of Algorithm 3 is passed to the pseudocode of Algorithm 4 as the set of vertices that will be used as the initial prefix Π, i.e. $\Pi = S'$. The best known solution Π^* is the current solution S, and the initial lower bound $LB = |S|$.

We recall that two restrictions were used to guarantee the uniqueness of a permutation in the enumeration tree generated by the original QClique algorithm, allowing to speedup the search by pruning. However, these restrictions might exclude some high quality solutions when the search starts from a set of fixed vertices.

In order to avoid the exclusion of such solutions, the uniqueness conditions are replaced by the requirement that $dens(G(\Pi)) \geq dens(G(\Pi'))$ in the $getCandidates$ method that defines the vertices that may be used to extend the current solution Π, ensuring that the conditions of Theorem 1 in (Ribeiro and Riveaux, 2017) are met to form a new solution Π'. In addition, as an strategy to improve the search process, the candidate vertices are taken in the non increasing order of their number of neighbors in the current solution. In case of ties, a vertex with maximum degree goes first.

Algorithm 5 displays the pseudocode of the $getCandidates$ function that builds the candidate list CL formed by the vertices that can be added at the end of a prefix solution. Line 1 initializes CL as empty and line 2 initializes i as the size of the current permutation Π. The loop in lines 3 to 8 considers the addition to the current permutation Π of all vertices in $V \setminus \Pi$. We denote by $deg_{G(\Pi)}(j)$ the number of vertices of the current permutation Π that are adjacent to vertex j. Line 4 checks if, for a candidate vertex j, the subgraph induced by $\Pi' = \Pi \cup \{j\}$ is a γ-clique and the density of the induced subgraph $G(\Pi')$

is not greater than that of $G(\Pi)$. If this is the case, then vertex j is definitely added to the candidate list CL in line 6. The vertices in the candidate list CL are sorted in the non increasing order of their number of neighbors in the current solution in line 9 and returned in line 10.

Algorithm 5 *getCandidates*(G, Π, γ)

1: $CL \leftarrow \emptyset$
2: $i \leftarrow |\Pi|$
3: **for all** $j \in V \setminus \Pi$ **do**
4: $\quad \Pi' \leftarrow \Pi \oplus \{j\}$
5: $\quad$ **if** $deg_{G(\Pi)}(j) + |E(\Pi)| \geq \gamma(i+1)i/2$ and $dens(G(\Pi)) \geq dens(G(\Pi'))$ **then**
6: $\quad\quad CL \leftarrow CL \cup \{j\}$
7: $\quad$ **end if**
8: **end for**
9: **sort** vertices in the candidate list CL by the non-decreasing order of their number of neighbors
10: **return** CL

6. Computational results

All algorithms were 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.

We have used 96 instances derived from maximum clique problems of the Second DIMACS Implementation Challenge (Johnson, 1996).

The newly proposed algorithm BRKGA-ExactQClique was compared with the original BRKGA-IG* heuristic of Pinto et al. (2017), 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 by Pinto et al. (2017) using the automatic tuning tool IRACE (López-Ibánez et al., 2011; Pérez Cáceres et al., 2014). The same settings were used for algorithm BRKGA-ExactQClique. Parameters settings for both algorithm are shown in Table 1. A parameter *maxnodes* is used as a maximum limit to the number of nodes of the search tree generated by the exact algorithm QClique. In case the search tree generated by the QClique algorithm reaches the maximum limit *maxnodes* of nodes, then it returns the best solution found until this point. The additional application of the DECODER-HCB reconstruction in line 13 of Algorithm 3 is used to further improve the current solution.

The number of vertices deleted in the destruction phase of DECODER-IG* of Algorithm 2 depends on the value of a parameter $\delta \in [0, 1]$. In order to to enforce that the fraction of nodes eliminated in the destruction phase is smaller for dense graphs, we empirically set $\delta = 1 - dens(G)$ if $dens(G) < 0.8$; $\delta = 2(1 - dens(G))$ otherwise . We note that no vertices will be removed if $dens(G) = 1$, i.e G is a clique (or a complete graph). However, in this case, the condition in line 10 of Algorithm 1 is true for every $S \subset V$ and every $v \in V \setminus S$. Therefore, in this case DECODER-HCB always finds the optimal solution $S = V$, which is the optimum for every complete graph G.

Tables 2 to 4 display average results over ten runs of each algorithm for each instance. A target value

Table 1: Parameters settings.

Algorithm	p	p_e	p_m	$rhoe$	α	δ	β	*maxnodes*	ρ
BRKGA-IG*	89	0.16	0.11	0.77	0.01	0.34	0.10	-	-
BRKGA-ExactQClique	89	0.16	0.11	0.77	0.01	-	0.10	131	0.12

is given to each instance. Each run stops when a solution at least as good as the target is found or a time limit of ten minutes is reached. For each instance, the table presents the threshold γ, the target for the size of the γ-clique, the average clique size found by algorithm BRKGA-IG* and the number of runs it matched or improved the target over the ten runs, the average clique size found by algorithm BRKGA-ExactQClique and the number of runs it matched or improved the target over the ten runs, and the average running time of each algorithm in seconds over the ten runs. Cells highlighted in boldface indicate the algorithms that attained the best values for each instance.

We observe in these tables that BRKGA-ExactQClique found strictly larger average γ-cliques than BRKGA-IG* for 28 instances, while BRKGA-IG* did better than BRKGA-ExactQClique in only ten instances. Regarding the number of runs for which each algorithm matched or improved the target, BRKGA-ExactQClique did better than BRKGA-IG* in 12 instances, while BRKGA-IG* performed better than BRKGA-ExactQClique in only two instances. Although BRKGA-ExactQClique clearly outperformed BRKGA-IG* in terms of solution quality, the latter was faster in 55 out of the 96 test instances.

In the next experiment, we evaluate and compare the run time distributions (or time-to-target plots – or ttt-plots, for short) of algorithms BRKGA-IG* and BRKGA-ExactQClique. 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 (1998) as a way to characterize the running times of stochastic local search algorithms for combinatorial optimization. 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 previous experiment and reported in Tables 2 to 4. The heuristics were run 200 times each, with different initial seeds for the pseudo-random number generator. Next, the empirical probability distributions of the time taken by each heuristic to find a target solution value are plotted. To plot the empirical distribution for each heuristic, we followed the methodology proposed by Aiex et al. Aiex et al. (2002, 2007). 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 3 to 6 illustrate the time-to-target plots for instances DSJC500.5, frb45-21-1, frb30-15-2, and frb30-15-5. These plots show that BRKGA-ExactQClique performed better for the two first instances, while BRKGA-IG* performed better for the two last ones. These conclusions are consistent with the results observed for each algorithm over the 200 runs that generated the time to target plots, as depicted in Table 5. This table shows that even though algorithm BRKGA-ExactQClique matches or improves BRKGA-IG* in terms of the average solution quality, the latter is faster than the former in terms of their average running times.

Figures 7 and 8 illustrate the evolution of the solution population along 100 generations of BRKGA-

Table 2: Numerical results over ten runs of each algorithm for each instance - Part A.

Instances	γ	target size	IG*	#opt.	QClique	#opt.	IG* (seconds)	QClique (seconds)
C125.9	0.999	34	**34.00**	**10**	**34.00**	**10**	**0.052**	0.207
C250.9	0.999	44	**44.00**	**10**	**44.00**	**10**	**0.213**	0.987
C500.9	0.999	57	**57.00**	**10**	**57.00**	**10**	**7.717**	41.596
C1000.9	0.999	67	**67.00**	**10**	66.80	8	**149.145**	181.719
C2000.9	0.999	74	73.60	6	**74.10**	**7**	456.268	**369.326**
C4000.5	0.8	46	43.20	0	**47.50**	**10**	648.695	**220.310**
DSJC500.5	0.8	34	**34.00**	**10**	**34.00**	**10**	22.445	**16.735**
DSJC1000.5	0.8	38	**38.10**	**10**	**38.10**	**10**	67.373	**63.808**
MANN_a9	0.999	16	**16.00**	**10**	**16.00**	**10**	**0.013**	0.023
MANN_a27	0.999	133	**133.00**	**10**	**133.00**	**10**	**1.869**	14.940
MANN_a45	0.999	428	427.20	6	**427.60**	**8**	360.498	**324.388**
brock200_1	0.8	114	**114.00**	**10**	**114.00**	**10**	**0.511**	1.211
brock200_2	0.8	24	**24.00**	**10**	**24.00**	**10**	1.027	**1.003**
brock200_3	0.8	41	**41.00**	**10**	**41.00**	**10**	**0.452**	0.964
brock400_1	0.8	189	**189.00**	**10**	**189.00**	**10**	**5.488**	12.592
brock400_2	0.8	186	**186.00**	**10**	**186.00**	**10**	**6.822**	12.548
brock400_3	0.8	187	**187.00**	**10**	**187.00**	**10**	40.283	**31.571**
brock800_1	0.8	94	93.70	8	**94.40**	**10**	163.726	**38.846**
brock800_2	0.8	93	93.00	**10**	**93.10**	**10**	99.053	**41.638**
brock800_3	0.8	92	**92.00**	**10**	**92.00**	**10**	118.929	**63.718**
c-fat200-1	0.5	30	**30.00**	**10**	**30.00**	**10**	**0.032**	0.310
c-fat200-2	0.5	58	**58.00**	**10**	**58.00**	**10**	**0.052**	0.544
c-fat200-5	0.5	148	**148.00**	**10**	**148.00**	**10**	**0.126**	0.898
c-fat500-1	0.5	35	**35.00**	**10**	**35.00**	**10**	**0.069**	0.548
c-fat500-2	0.5	66	**66.00**	**10**	**66.00**	**10**	**0.135**	1.246
c-fat500-5	0.5	164	**164.00**	**10**	**164.00**	**10**	**0.311**	2.625
c-fat500-10	0.5	324	**324.00**	**10**	**324.00**	**10**	**0.725**	3.960
frb59-26-5	0.95	216	216.60	7	**216.80**	**10**	422.615	**224.426**
frb59-26-4	0.95	222	222.60	9	**223.40**	**10**	273.719	**149.048**
frb59-26-2	0.95	229	229.00	8	**229.40**	**10**	372.323	**127.098**
frb59-26-1	0.95	232	232.10	8	**233.00**	**10**	466.144	**152.021**

IG* and BRKGA-ExactQClique for one execution of instances frb59-26-2 and frb50-23-5, respectively. In addition, Figures 9 and 10 display how the best solutions found by the two algorithms evolve along the first 1000 seconds of processing time, for the same instances frb59-26-2 and frb50-23-5, respectively. They show that BRKGA-ExactQClique systematically finds better solutions faster than BRKGA-IG*. The best solution value obtained by BRKGA-ExactQClique is better than or equal to that found by

Table 3: Numerical results over ten runs of each algorithm for each instance - Part B.

Instances	γ	target size	IG*	#opt.	QClique	#opt.	IG* (seconds)	QClique (seconds)
frb56-25-5	0.95	192	192.30	**10**	**193.50**	**10**	230.380	**76.185**
frb56-25-4	0.95	190	189.00	4	**190.40**	**9**	502.663	**285.087**
frb56-25-2	0.95	204	204.50	**10**	**204.80**	**10**	290.945	**119.251**
frb56-25-1	0.95	220	221.00	**10**	**220.20**	**10**	295.512	**150.905**
frb53-24-5	0.95	162	162.50	**10**	**162.80**	**10**	225.562	**105.855**
frb53-24-4	0.95	174	175.00	9	**174.50**	**10**	233.042	**167.407**
frb53-24-2	0.95	168	168.00	8	**168.20**	**10**	398.870	**356.155**
frb53-24-1	0.95	192	192.90	8	**193.10**	**10**	287.673	**143.790**
frb50-23-5	0.95	156	**156.30**	9	156.00	**10**	358.773	**183.970**
frb50-23-4	0.95	150	**150.80**	**10**	150.30	**10**	125.065	**45.791**
frb50-23-2	0.95	153	152.90	8	**153.30**	**10**	355.328	**101.622**
frb50-23-1	0.95	153	153.80	**10**	**154.20**	**10**	160.822	**98.673**
frb45-21-5	0.95	118	**118.60**	**10**	118.40	**10**	111.796	**53.902**
frb45-21-4	0.95	125	**125.40**	**10**	125.20	**10**	47.415	**36.153**
frb45-21-2	0.95	120	**120.30**	**10**	120.10	**10**	98.871	**61.110**
frb45-21-1	0.95	119	119.00	**10**	**119.80**	**10**	59.476	**18.847**
frb40-19-5	0.95	98	98.10	**10**	**98.40**	**10**	56.161	**48.600**
frb40-19-4	0.95	94	94.10	**10**	**94.30**	**10**	34.232	**20.139**
frb40-19-2	0.95	101	101.10	**10**	**101.20**	**10**	72.465	**56.743**
frb40-19-1	0.95	109	**109.20**	**10**	109.10	**10**	31.920	**22.767**
frb35-17-5	0.95	78	78.00	**10**	**78.30**	**10**	**19.124**	22.494
frb35-17-4	0.95	79	**79.20**	**10**	**79.20**	**10**	**19.196**	43.029
frb35-17-2	0.95	73	**73.30**	**10**	73.20	**10**	**7.134**	9.398
frb35-17-1	0.95	77	**77.20**	**10**	77.00	**10**	**18.130**	18.475
frb30-15-5	0.95	59	**59.00**	**10**	**59.00**	**10**	6.447	**5.907**
frb30-15-4	0.95	60	**60.00**	**10**	**60.00**	**10**	**44.322**	135.412
frb30-15-2	0.95	58	**58.00**	**10**	**58.00**	**10**	**5.778**	9.608
frb30-15-1	0.95	59	**59.00**	**10**	**59.00**	**10**	7.652	**5.563**
gen200_p0.9_44	0.999	40	**40.00**	**10**	**40.00**	**10**	**0.114**	0.617
gen400_p0.9_55	0.999	53	**53.00**	**10**	**53.00**	**10**	**8.169**	103.434
gen400_p0.9_65	0.999	58	**58.50**	**10**	57.50	6	**45.234**	379.328

BRKGA-IG* anytime along the runs displayed in these figures.

Table 4: Numerical results over ten runs of each algorithm for each instance - Part C.

Instances	γ	target size	IG*	#opt.	QClique	#opt.	IG* (seconds)	QClique (seconds)
hamming6-2	0.95	37	**37.00**	**10**	**37.00**	**10**	**0.033**	0.073
hamming6-4	0.5	32	**32.00**	**10**	**32.00**	**10**	**0.028**	0.163
hamming8-2	0.999	129	**129.00**	**10**	**129.00**	**10**	0.233	**0.214**
hamming8-4	0.8	71	**71.00**	**10**	**71.00**	**10**	**0.341**	0.798
hamming10-2	0.999	525	**525.00**	**10**	**525.00**	**10**	12.910	**7.591**
hamming10-4	0.95	82	82.00	**10**	**82.50**	**10**	48.831	**11.117**
johnson8-4-4	0.8	43	**43.00**	**10**	**43.00**	**10**	0.056	**0.310**
johnson16-2-4	0.8	34	**34.00**	**10**	**34.00**	**10**	**0.061**	0.195
johnson32-2-4	0.95	21	**21.00**	**10**	**21.00**	**10**	**0.704**	0.909
keller4	0.8	54	**54.00**	**10**	**54.00**	**10**	**0.178**	0.578
keller5	0.8	486	**486.00**	**10**	**486.00**	**10**	**4.467**	15.628
keller6	0.95	271	271.10	8	**272.00**	**9**	416.216	**388.637**
p_hat300-1	0.5	64	**64.00**	**10**	**64.00**	**10**	**2.604**	36.551
p_hat300-2	0.8	114	**114.00**	**10**	**114.00**	**10**	**0.927**	2.000
p_hat500-1	0.5	96	**96.00**	**10**	**96.00**	**10**	**1.916**	6.745
p_hat500-2	0.8	211	**211.00**	**10**	**211.00**	**10**	**1.865**	4.451
p_hat700-1	0.5	119	**119.00**	**10**	**119.00**	**10**	**13.095**	55.259
p_hat700-2	0.8	288	**288.00**	**10**	**288.00**	**10**	**3.695**	8.318
p_hat1000-1	0.5	144	144.00	**10**	**144.10**	**10**	**14.248**	32.708
p_hat1000-2	0.8	385	**385.00**	**10**	**385.00**	**10**	**9.384**	20.052
p_hat1000-3	0.95	210	**210.00**	**10**	**210.00**	**10**	**10.737**	15.623
p_hat1500-2	0.8	642	**642.00**	**10**	**642.00**	**10**	**31.317**	53.467
san1000	0.8	562	**562.00**	**10**	**562.00**	**10**	**10.101**	16.393
san200_0.7_1	0.95	57	**57.00**	**10**	**57.00**	**10**	**0.195**	0.444
san200_0.7_2	0.95	34	**34.00**	**10**	**34.00**	**10**	**0.206**	0.520
san200_0.9_1	0.999	55	55.70	**10**	**55.50**	**10**	**6.079**	24.426
san200_0.9_2	0.999	60	**60.00**	**10**	**60.00**	**10**	**0.410**	1.323
san200_0.9_3	0.999	42	**42.20**	**10**	**42.20**	**10**	**12.199**	49.317
san400_0.5_1	0.6	285	**285.00**	**10**	**285.00**	**10**	**1.236**	3.239
san400_0.7_1	0.95	201	**201.00**	**10**	**201.00**	**10**	**1.137**	1.885
san400_0.7_2	0.95	62	**62.00**	**10**	**62.00**	**10**	**0.566**	0.903
san400_0.7_3	0.95	40	**40.00**	**10**	**40.00**	**10**	**25.685**	52.996
sanr400_0.7	0.95	32	**32.00**	**10**	**32.00**	**10**	**2.130**	2.486
sanr400_0.5	0.8	32	**32.00**	**10**	**32.00**	**10**	**1.527**	1.989

7. Concluding remarks

We proposed an improvement in the biased random-key genetic algorithm BRKGA-IG* of Pinto et al. (2017) for approximately solving the maximum quasi-clique problem, hybridizing a local search al-

Fig. 3: Time to target plot for instance DSJC500.5.

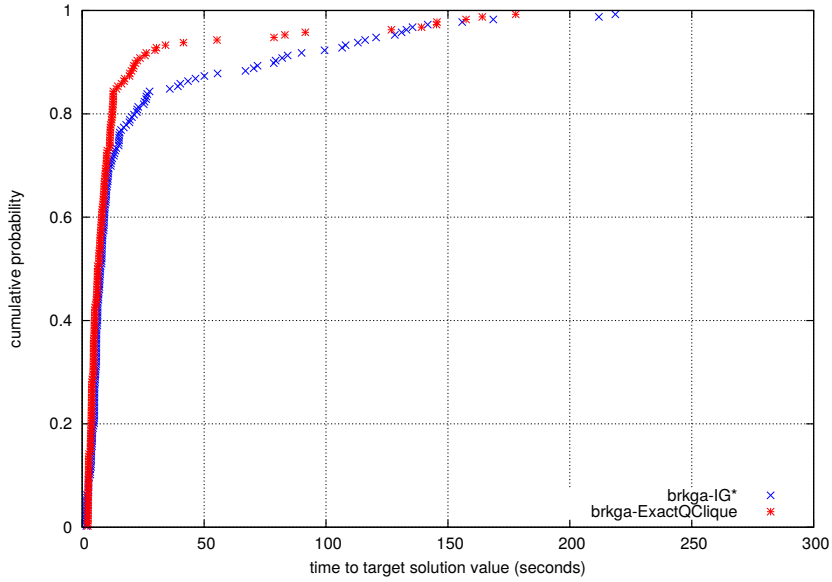


Fig. 4: Time to target plot for instance frb45-21-1.

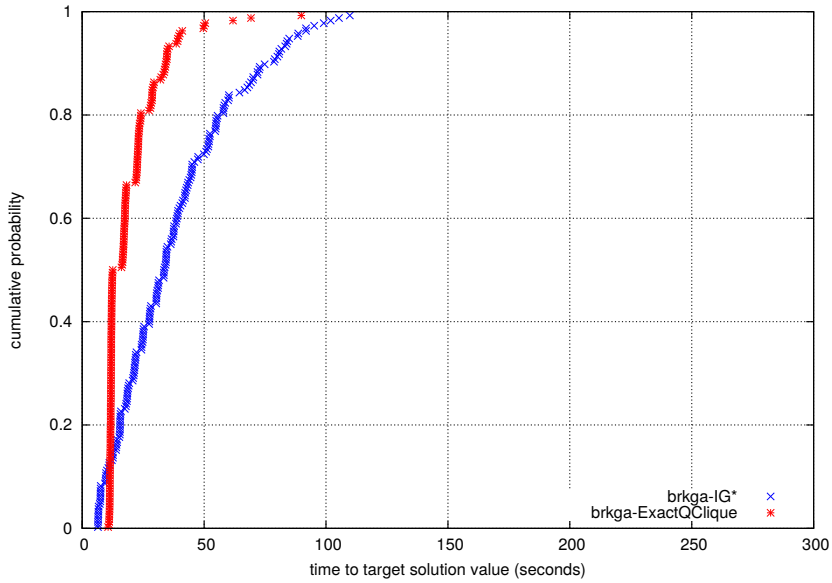


Fig. 5: Time to target plot for instance frb30-15-2.

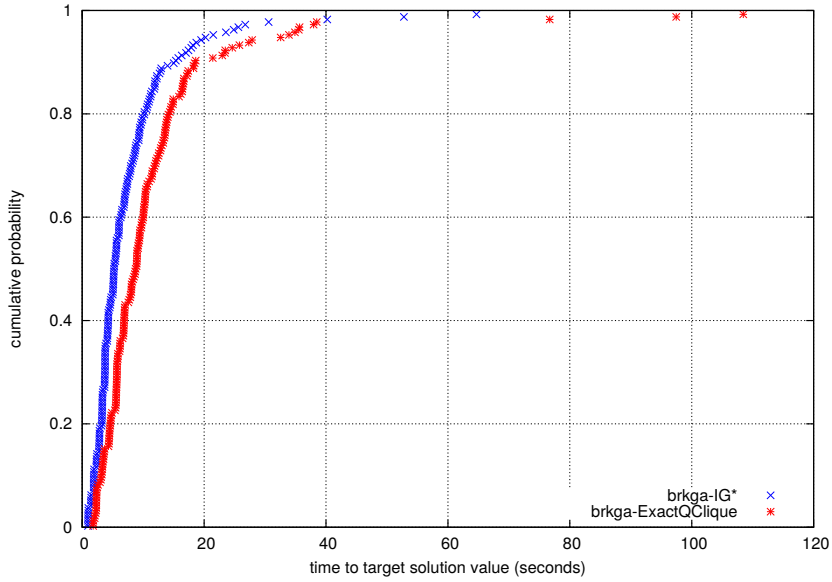


Fig. 6: Time to target plot for instance frb30-15-5.

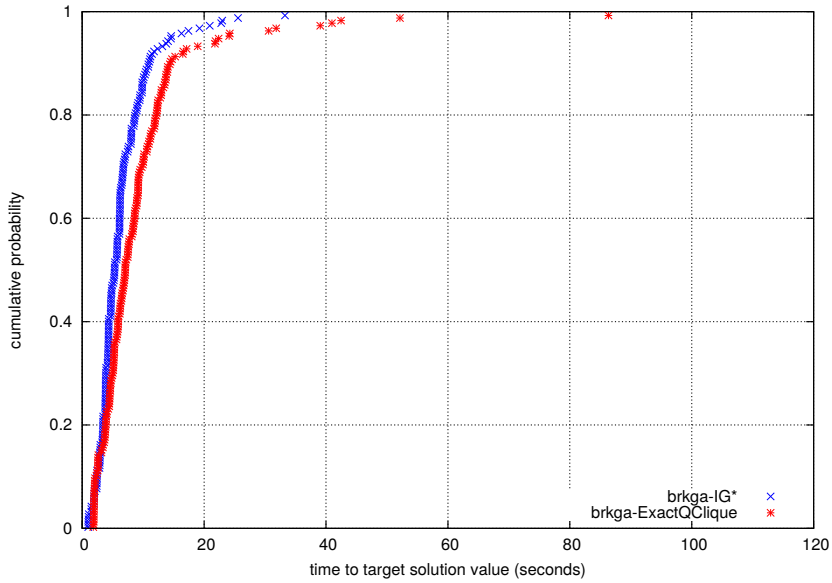
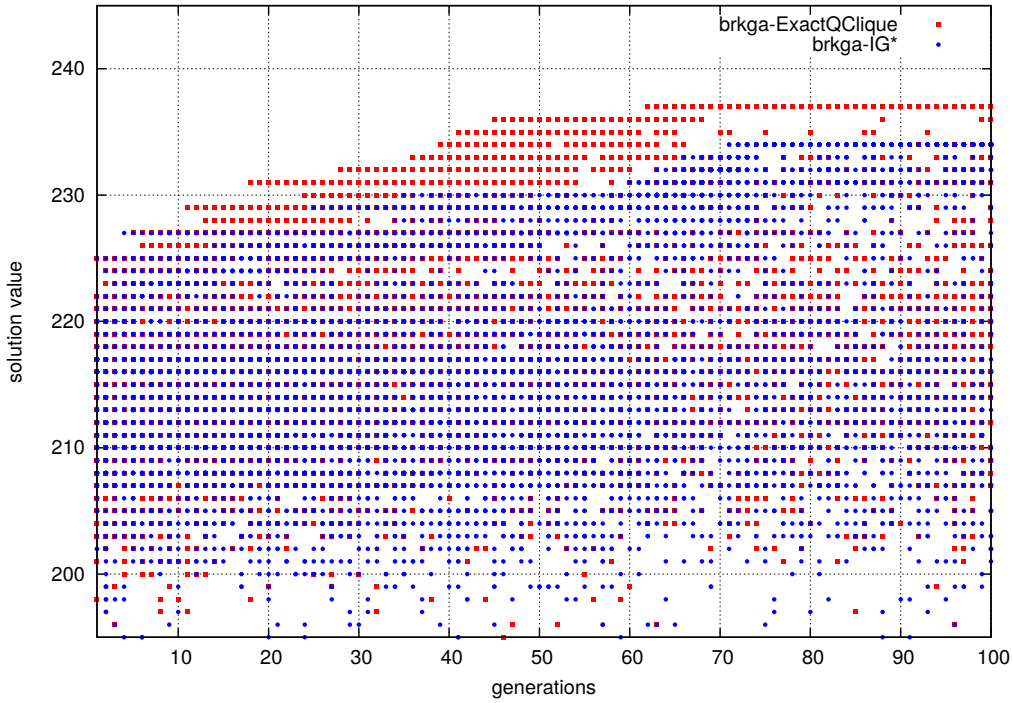


Table 5: Numerical results over 200 runs of each algorithm for each instance.

Instances	γ	target size	IG*	#opt.	QClique	#opt.	IG* (seconds)	QClique (seconds)
DSJC500.5	34	0.8	**34.000**	**200**	**34.000**	**200**	22.048	**14.376**
frb45-21-1	119	0.95	119.325	**200**	**119.525**	**200**	37.418	**19.143**
frb30-15-2	58	0.95	**58.000**	**200**	**58.000**	**200**	**7.477**	11.301
frb30-15-5	59	0.95	59.010	**200**	**59.055**	**200**	**6.350**	9.046

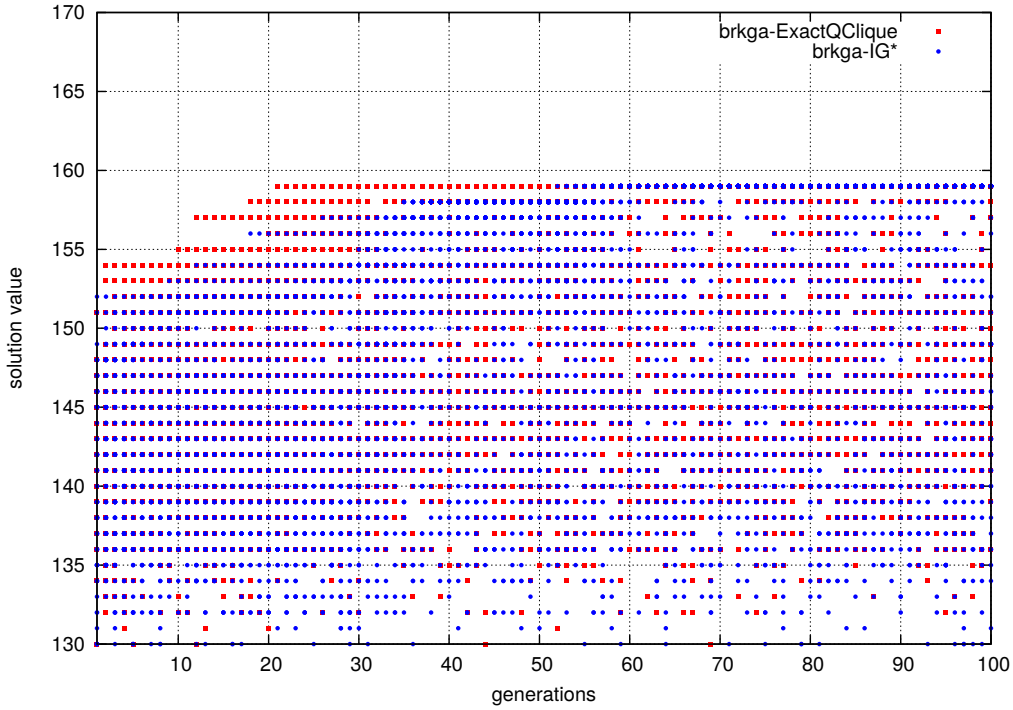
Fig. 7: Population evolution for instance frb59-26-2 along 100 generations of BRKGA-IG* and BRKGA-ExactQClique.



gorithm that is based on an exact algorithm. The exact local search is a modification of the QClique algorithm of Ribeiro and Riveaux (2017). The numerical results showed that the new, hybrid approach outperforms BRKGA-IG*, finding better solutions for most test problems.

Future work will focus on improvements of the exact local search method and better parameter settings. In particular, we are interested in investigating better values for the destruction parameter δ and for

Fig. 8: Population evolution for instance frb50-23-5 along 100 generations of BRKGA-IG* and BRKGA-ExactQClique.



the maximum number of nodes $maxnodes$ in the search tree. The exact algorithm may also be further improved by cuts that prune duplicate solutions.

References

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*, Springer, pp. 598–612.

Aiex, R., Resende, M., Ribeiro, C.C., 2002. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics* 8, 343–373.

Aiex, R., Resende, M., Ribeiro, C.C., 2007. TTTPLOTS: A Perl program to create time-to-target plots. *Optimization Letters* 1, 355–366.

Bean, J.C., 1994. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing* 2, 154–160.

Brandão, J.S., Noronha, T.F., Resende, M.G.C., Ribeiro, C.C., 2015. A biased random-key genetic algorithm for single-round divisible load scheduling. *International Transactions Operational Research* 22, 823–839.

Brandão, J.S., Noronha, T.F., Resende, M.G.C., Ribeiro, C.C., 2017. A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems. *International Transactions Operational Research* 27, 1061–1077.

Fig. 9: Evolution of the best value found by BRKGA-IG* and BRKGA-ExactQClique along the 1000 first seconds of running time for instance frb59-26-2.

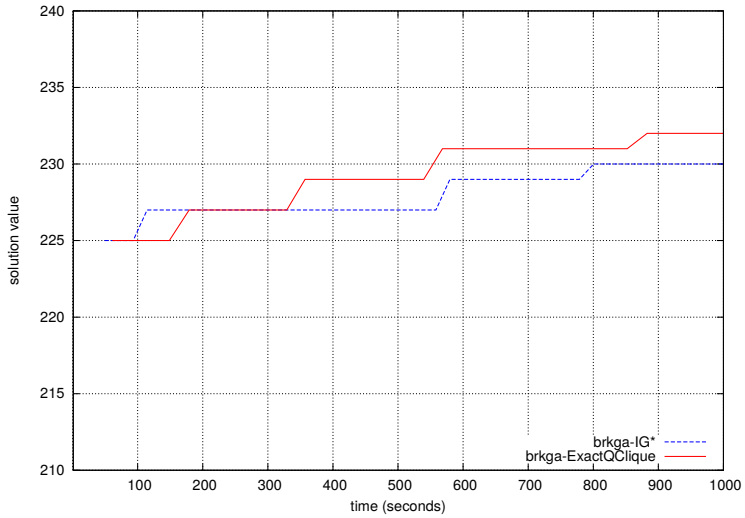
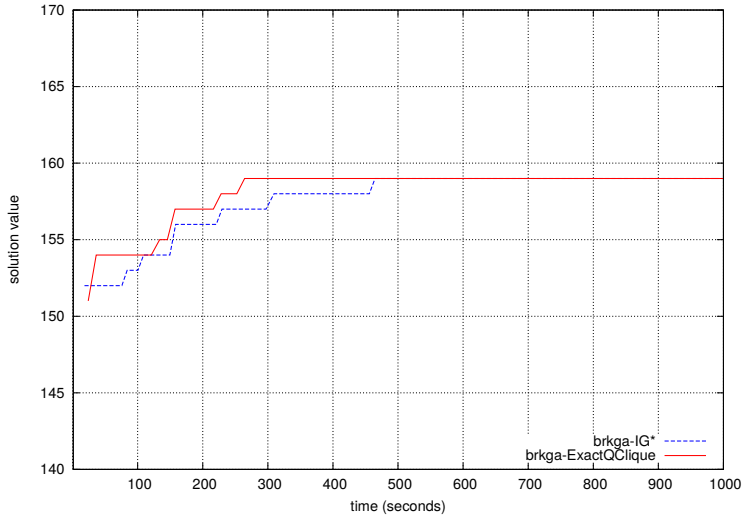


Fig. 10: Evolution of the best value found by BRKGA-IG* and BRKGA-ExactQClique along the 1000 first seconds of running time for instance frb50-23-5.



Brunato, M., Hoos, H., Battiti, R., 2008. On effectively finding maximal quasi-cliques in graphs. In Maniezzo, V., Battiti, R. and Watson, J.P. (eds), *Learning and Intelligent Optimization*, *Lecture Notes in Computer Science*. Vol. 5313. Springer, Berlin, pp. 41–55.

FICO, 2017. FICO Xpress Optimization Suite 7.6. http://www.fico.com/en/products/fico-xpress-optimization-suite. Online reference, last visited on May 25, 2017.

Goncalves, J.F., Resende, M.G.C., 2011. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics* 17, 487–525.

Goncalves, J.F., Resende, M.G.C., Toso, R.F., 2013. Biased and unbiased random key genetic algorithms: An experimental analysis. In *Abstracts of the 10th Metaheuristics International Conference*, Singapore.

Hoos, H., Stützle, T., 1998. Evaluation of Las Vegas algorithms - Pitfalls and remedies. In Cooper, G. and Moral, S. (eds), *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, Madison, pp. 238–245.

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.

Karp, R.M., 1972. Reducibility among combinatorial problems. In Miller, R.E. and Thatcher, J.W. (eds), *Complexity of Computer Computations*, Plenum, New York, pp. 85–103.

López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M., 2011. The IRACE package: Iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium.

Noronha, T.F., Resende, M.G.C., Ribeiro, C.C., 2011. A biased random-key genetic algorithm for routing and wavelength assignment. *Journal of Global Optimization* 50, 503–518.

Oliveira, A.B., 2013. Heurísticas para o Problema de Quasi-Clique de Cardinalidade Máxima. Master's thesis, Universidade Federal Fluminense, Niterói, Brazil.

Oliveira, A.B., Plastino, A., Ribeiro, C.C., 2013. Construction heuristics for the maximum cardinality quasi-clique problem. In *Abstracts of the 10th Metaheuristics International Conference*, Singapore, p. 84.

Pajouh, F.M., Miao, Z., Balasundaram, B., 2014. A branch-and-bound approach for maximum quasi-cliques. *Annals of Operations Research* 216, 1, 145–161.

Pattillo, J., Veremyev, A., Butenko, S., Boginski, V., 2013. On the maximum quasi-clique problem. *Discrete Applied Mathematics* 161, 244–257.

Pérez Cáceres, L., López-Ibáñez, M., Stützle, T., 2014. An analysis of parameters of IRACE. In *Proceedings of the 14th European Conference on Evolutionary Computation in Combinatorial Optimization*, *Lecture Notes in Computer Science*. Vol. 8600. Springer, Berlin, pp. 37–48.

Pinto, B.Q., Ribeiro, C.C., Riveaux, J.A., Rosseti, I., 2017. A biased random-key genetic algorithm for solving the maximum quasi-clique problem. *Journal of Global Optimization* Submitted.

Ribeiro, C.C., Riveaux, J.A., 2017. An exact algorithm for the maximum quasi-clique problem. *Journal of Global Optimization* Submitted.

Spears, W., de Jong, K., 1991. On the virtues of parameterized uniform crossover. In Belew, R. and Booker, L. (eds), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufman, San Mateo, pp. 230–236.

Toso, R.F., Resende, M.G.C., 2015. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software* 30, 81–93.

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, 1, 177–214.

ANEXO C – Pinto, B. Q., Ribeiro, C. C., Rosseti, I., Noronha, T. F. "A biased random-key genetic algorithm for routing and wavelength assignment under a sliding scheduled traffic model". Submetido para Journal of Global Optimization.

Noname manuscript No.
(will be inserted by the editor)

A biased random-key genetic algorithm for routing and wavelength assignment under a sliding scheduled traffic model

Bruno Q. Pinto · Celso C. Ribeiro · Isabel Rosseti · Thiago F. Noronha

Received: December 2018

Abstract The problem of routing and wavelength assignment in optical networks consists in minimizing the number of wavelengths that are needed to route a set of demands, such that demands routed using lightpaths that share common links are assigned to different wavelengths. We present a biased random-key genetic algorithm for approximately solving the problem of routing and wavelength assignment of sliding scheduled lightpath demands in optical networks. In this problem variant, each demand is characterized not only by a source and a destination, but also by a duration and a time window in which it has to be met. Computational experiments show that the numerical results obtained by the proposed heuristic improved upon those obtained by a multistart constructive heuristic. In addition, the biased random-key genetic algorithm obtained much better results than an existing algorithm for the problem, finding solutions that use roughly 50% of the number of wavelengths determined by the latter.

Keywords biased random-key genetic algorithm · metaheuristics · routing and wavelength assignment · sliding scheduled lightpath demands · scheduled lightpath demands · optical networks

Bruno Q. Pinto
Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro, Uberlândia, MG 38411-104, Brazil.
E-mail: bruno.queiroz@iftm.edu.br

Celso C. Ribeiro
Universidade Federal Fluminense, Institute of Computing, Niterói, RJ 24210-240, Brazil.
E-mail: celso@ic.uff.br

Isabel Rosseti
Universidade Federal Fluminense, Institute of Computing, Niterói, RJ 24210-240, Brazil.
E-mail: rosseti@ic.uff.br

Thiago F. Noronha
Universidade Federal de Minas Gerais, Department of Computer Science, Belo Horizonte, MG 31270-901, Brazil.
E-mail: tfn@dcc.ufmg.br

1 Introduction

Given the physical topology of a WDM (Wavelength Division Multiplexing) optical network and a set of lightpath demands over this network, the problem of routing and wavelength assignment (RWA) consists in routing the lightpaths and assigning a wavelength to each of them. Lightpaths that share common links must necessarily be assigned to different wavelengths.

The demands for lightpaths are known in advance. Every lightpath must be assigned to the same wavelength along all links that form its route, since there is no wavelength conversion. There may be an arbitrary demand for lightpaths between any pair of nodes, since the traffic between them can be larger than the maximum capacity supported by a single lightpath.

Variants of this problem are characterized by different optimization criteria and traffic patterns, see e.g. [5,10,22,30,28,31,34,46,47]. The objective of the min-RWA problem variant [12,30] is to minimize the number of wavelengths that are needed to establish all lightpath requests. The problem was determined to be NP-hard in [12]. Heuristics for min-RWA appear e.g. in [5,28,30, 16,23,26,42].

Routing and wavelength assignment may not only consider static traffic models, but also dynamic and scheduled models. In the case of scheduled traffic models, each demand for lightpaths requires the use of the network over some specific time period. Two problem variants exist: (a) scheduled lightpath demands (RWA-SLD) [22,41] and (b) sliding scheduled lightpath demands (RWA-SSLD) [4,18–20,24,39,40], following its introduction by Wang et al. [45,46] in the context of service provisioning under a scheduled traffic model in reconfigurable optical networks. In the former, the demands must be assigned to the resources of the network at the specific times where they will be needed [22]. In the latter, only a time window longer than the duration of each demand is given [46].

According to Kuri et al. [22], scheduled traffic models are more realistic due to the periodic nature of data traffic, which is not homogeneous along the day. We consider the RWA-SSLD problem variant in this work, assuming that the resources of the network (capacity and wavelengths) are unbounded and minimizing the number of wavelengths that are needed to route all demands for lightpaths within the time windows assigned to each of them.

In the following, we propose and describe a biased random-key genetic algorithm (BRKGA) for routing and wavelength assignment under a sliding scheduled traffic model. The remainder of this article is organized as follows. The sliding scheduled lightpath demands problem (RWA-SSLD) and related work are presented in the next section. Section 3 gives an overview of BRKGAs and their customization to the sliding scheduled lightpath demands problem. Section 4 describes a constructive heuristic and a decoder. A test instance generator and computational experiments are reported in Section 5. The numerical results obtained by the proposed approach for approximately solving problem RWA-SSLD improved upon those obtained by a multistart constructive heuristic. We also report a comparison of the results obtained by the

biased random-key genetic algorithm with those obtained by Liu et al. [24], reproducing as closely as possible the same conditions tested in their experiments. The new heuristic outperformed the algorithm in the literature, finding solutions that use roughly 50% of the number of wavelengths determined by the latter. Concluding remarks are presented in the final section.

2 Routing and wavelength assignment under a sliding scheduled traffic model

2.1 Scheduled lightpath demands problem (RWA-SLD)

Let $G = (V, E)$ be a graph representing the network topology, where V represents the set of the nodes and E denotes the set of edges or links connecting the nodes. Let also $D = \{d_1, \ldots, d_m\}$ be a set of m demands for lighpaths, where each demand d_i is defined by a source s_i, a destination t_i, a number n_i of requested lightpaths, a starting time a_i and a finishing time b_i, for $i = 1, \ldots, m$.

Kuri et al. [22] established the problem of minimizing the number of wavelenghts needed to route all lightpath requests at their specific starting and finishing times. For each demand d_i, all n_i lightpaths should be routed along the same links using n_i different wavelenghts, for $i = 1, \ldots, m$. Skorin-Kapov [41] proposed heuristics for approximately solving RWA-SLD. Among them, DP_RWA_SLD is an adaptive greedy heuristic based on the algorithm proposed in [26] for solving problem min-RWA.

2.1.1 Heuristic DP-RWA-SLD

Algorithm 1 describes the greedy heuristic DP_RWA_SLD originally proposed by Skorin-Kapov [41] for solving RWA-SLD, based on that of Manohar et al. [26] for the min-RWA problem. Each of its iterations determines a new subset of non-conflicting demands that can be routed using different wavelengths over disjoint paths in the network.

Heuristic DP_RWA_SLD receives as inputs the graph $G = (V, E)$ representing the network and the set $D = D_{SLD}$ of demands $d_1, \ldots, d_m$ in non-increasing order of their number of requested wavelengths. Its outputs are the number λ of wavelengths needed to route all demands, as well as the set W_d of wavelengths that may be used by demand d and the set of edges $Path_d$ in the route assigned to demand d, for every $d \in D$.

Let $c_{s,t}(G)$ and $R_{s,t}(G)$ denote, respectively, the number and the set of edges in the shortest path from node s to t in $G = (V, E)$. The diameter of graph G is $diam(G) = \max\{c_{s,t}(G) : s, t \in V\}$.

The number λ of wavelengths needed to route all demands is set to zero in line 1. The loop in lines 2–5 initializes sets W_d and $Path_d$, for every demand $d \in D$. The loop in lines 6–24 is performed until a wavelength has been assigned to every demand. Line 7 initializes as empty the subset of yet unassigned demands that can be routed without conflicts. The loop in lines 8–20 investigates all

yet unassigned demands $d \in D$. A temporary copy G' of graph G is built in line 9. The loop in lines 10–14 checks in line 11 if demand d overlaps in time with each demand $p \in Partition$ already routed. In case of overlapping, the edges in path $Path_p$ cannot be used to route demand d and must be removed from the set of candidate edges $E(G')$.

If the length $c_{s_d,t_d}(G')$ of the shortest path from s_d to t_d in G' is smaller than or equal to $\max\{diam(G), \sqrt{|E|}\}$, then demand d is inserted in $Partition$ in line 16 and the arcs in path $R_{s_d,t_d}(G')$ are saved in $Path_d$ in line 17. The wavelengths indexed by $\lambda + 1, \ldots, \lambda + n_d$ will be used to route demand d in line 18. The number λ of lightpaths needed is increased by $w = \max\{n_p : p \in Partition\}$ in line 21. The demands selected in $Partition$ that will be routed with the new selected wavelengths are removed from D in line 22. Line 24 returns the number of wavelengths needed, the wavelengths assigned to the demands, and the lightpaths used to route the demands.

Algorithm 1 DP_RWA_SLD($G, D = D_{SLD}$)

```
 1: λ ← 0
 2: for all d ∈ D do
 3:    W_d ← ∅
 4:    Path_d ← ∅
 5: end for
 6: while D ≠ ∅ do
 7:    Partition ← ∅
 8:    for all d ∈ D do
 9:       G' ← G
10:       for all p ∈ Partition do
11:          if a_d ≤ a_p < b_d or a_p ≤ a_d < b_p then
12:             E(G') ← E(G') \ Path_p
13:          end if
14:       end for
15:       if c_{s_d,t_d}(G') ≤ max{diam_G, √|E|} then
16:          Partition ← Partition ∪ {d}
17:          Path_d ← R_{s_d,t_d}(G')
18:          W_d ← {λ + 1, ..., λ + n_d}
19:       end if
20:    end for
21:    λ ← λ + max{n_p : p ∈ Partition}
22:    D ← D \ Partition
23: end while
24: return λ, (W_d, d = 1, ..., m), (Path_d, d = 1, ..., m)
```

2.2 Sliding scheduled lightpath demands problem (RWA-SSLD)

This variant originally described by Wang et al. [45,46] is a generalization of problem RWA-SLD, when time windows are defined for each lightpath requested, instead of starting and finishing times. Real applications are more likely to behave as RWA-SSLD problems, e.g. whenever a company desires

to optimize the cost of a network by enforcing traffic during times with less use for routine operations such as data back-up [46]. According to Jaekel et al. [20], such strategies allow for energy optimization.

Once again, we denote by $G = (V, E)$ a graph representing the network topology, where V and E are, respectively, the set of nodes and the set of edges. We also denote by $D = \{d_1, \ldots, d_m\}$ a set of m demands for lighpaths, where each demand d_i is defined by a source s_i, a destination t_i, a number n_i of requested lightpaths, a time window starting at a_i and finishing at b_i, and a duration τ_i, for $i = 1, \ldots, m$. The starting time f_i of the transmission of lightpath request d_i is not known beforehand but should satisfy the time window constraint $a_i \leq f_i \leq b_i - \tau_i$. Table 1 illustrates an example with $m = 4$ demands for the network in Figure 1.

Table 1 Example of problem RWA-SSLD with $m = 4$ demands for lighpaths.

Demands	s_i	t_i	n_i	a_i	b_i	τ_i
d_1	A	E	2	1	9	4
d_2	B	E	1	3	10	5
d_3	A	D	3	7	12	4
d_4	C	E	4	4	11	3

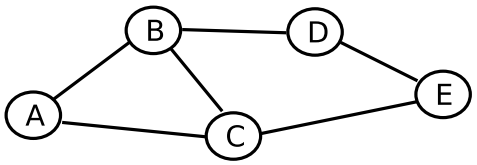


Fig. 1 Network topology example.

Figure 2 illustrates a time window starting at a_i and finishing at b_i for the scheduling of demand d_i with duration τ_i starting any time $f_i \in [a_i, b_i]$.

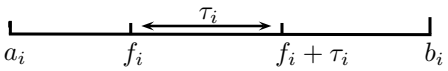


Fig. 2 Sliding time window model.

In the example of Figure 3, corresponding to task d_1 of Table 1, the starting time $f_1 = 3$ of the transmission of lightpath request d_1 can be anticipated or delayed by two time units.

If all time windows are tight, i.e., $b_i - a_i = \tau_i$, for $i = 1, \ldots, m$, then there are no time slacks and problems RWA-SSLD and RWA-SLD coincide.

For practical purposes, the length of each lightpath is bounded from above by $\max\{diam(G), \sqrt{|E|}\}$, where $diam(G)$ denotes the diameter of graph $G =$

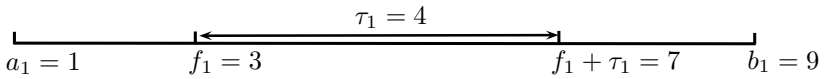


Fig. 3 Example of a sliding time window for demand d_1 of Table 1.

(V, E), i.e., the number of edges in the shortest path between the pair of nodes in V whose shortest path between them is maximum over all pairs of nodes in V [21].

2.3 Literature review

Wang et al. [45,46] introduced the problem of service provisioning on a sliding scheduled traffic model in optical networks. The approach proposed in [46] decomposes RWA-SSLD in two subproblems: (a) how to appropriately place a demand within its corresponding time window to reduce overlap in time over the set of demands; and (b) route and assign wavelengths to a set of demands under the sliding scheduled traffic model in mesh reconfigurable optical networks without wavelength conversion. Another algorithm is used to rearrange the demands that could not be satisfied by the routing and wavelength assignment algorithm by negotiating a new setup time. The primary objective of the proposed algorithms consists in scheduling the set of demands in such a way that the number of wavelengths used in the entire network is minimized when the network resources are sufficient to route all demands. If the network resources are not sufficient to route all demands, then the objective becomes to minimize the total number of wavelengths used, including those used by the demands rearranged. Network NSFNET with 14 nodes was used in the computational experiments, with the number of wavelengths that can be used in each edge limited to 30 and the number of demands $m \in [50, 400]$. The test problems considered three classes of demands: low, medium, and highly timely correlated demands [22].

Liu et al. [24] addressed the particular case of RWA-SSLD where $n_i = 1$, for $i = 1, \ldots, m$, using a partition coloring model. Network topologies USANET and NSFNET with 24 and 14 nodes, respectively, were used in the computational experiments, with the number of demands $m \in [10, 100]$. In this work, the length of any lightpath cannot be longer than the shortest path between its extremities by more than δ hops. The maximum-conflict degre-first conflict reducing algorithm (MCDF-CR) was compared with algorithm IPSR developed in [4]. The numerical results showed that on average MCDF-CR reduced the number of wavelenghts used by 9.9%.

Jaekel et al. [18] proposed integer programming formulations for both RWA-SLD and RWA-SSLD, minimizing the number of wavelengths used or the network congestion (measured in terms of the number of wavelengths on the most heavily loaded link). Networks NFSNET with 14 nodes and that in [17] with 10 nodes were used in the computational experiments, with the number of wavelengths that can be used in each edge limited to 32. Again, the

test problems considered three classes of demands: low, medium, and highly timely correlated demands [22]. The time windows had durations of 2, 4 or 6 hours.

Andrei et al. [4] proposed a new formulation for RWA-SSLD and a new heuristic named IPSR-LR (integrated provisioning of sliding requests). They assumed that all demands request the same number of wavelengths. Network USANET with 24 nodes was used in the computational experiments, with the number of demands $m \in [250, 1400]$.

Jaekel et al. [20] investigated variants of RWA-SSLD and RWA-SLD that minimize energy consumption and transmission cost over the network. They proposed integer programming formulation algorithms for problems RWA-SLD and RWA-SSLD, as well as a genetic algorithm for instances with up to 20 nodes and hundreds of demands. Networks NSFNET and ARPANET with 14 and 20 nodes, respectively, were used in the computational experiments, with the number of demands $m \in [60, 800]$. Once again following [22], the test problems included instances with three classes of demands: weakly, medium, and strongly time correlated.

In this work, we consider problem RWA-SSLD assuming that the resources of the network (capacity and wavelengths) are unbounded and minimizing the number of wavelengths that are needed to route all demands for lightpaths within the time windows assigned to each of them, using the notation introduced in Sections 2.1 and 2.2. A biased random-key genetic algorithm for RWA-SSLD is presented in the next section.

3 A biased random-key genetic algorithm for RWA-SSLD

Random-key genetic algorithms (RKGA) were introduced by Bean [6]. Solutions are associated with vectors of randomly generated real numbers called keys. A deterministic algorithm, which in this context is also called a decoder, takes a vector of real keys 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 repetions allowed.

A biased random-key genetic algorithm (BRKGA) differs from an RKGA by the strategy used to select parents for crossover, see Resende and Ribeiro [36] for an advanced tutorial of methods and applications. BRKGAs have been applied very successfully to a large variety of scheduling problems and, in particular, to simpler variants of routing and wavelength assignment (see, e.g., [10, 30]).

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 the offspring.

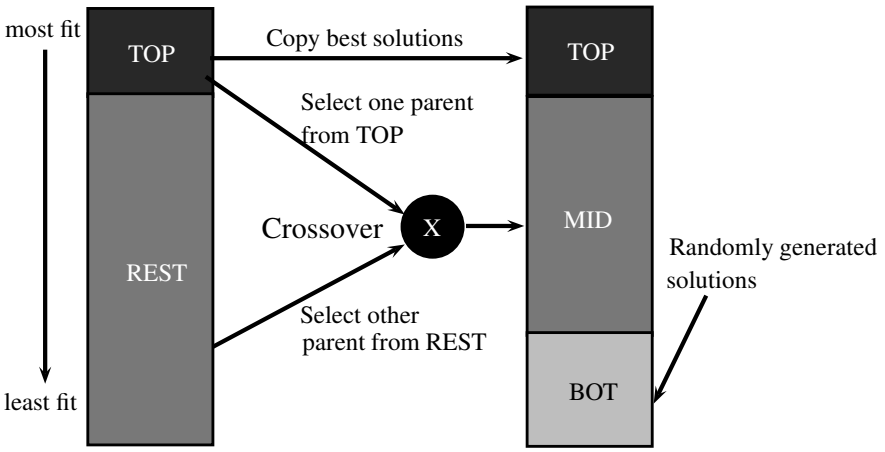


Fig. 4 Population evolution between consecutive generations of a BRKGA.

In this work, we develop a BRKGA for RWA-SSLD that evolves a population formed by vectors of real numbers. Each chromosome of this population is a vector of $|D|$ components. Each of these components, or keys, is a real number in the range $[0, 1)$, associated with one of the demands of the set D. The chromosome is decoded by an algorithm that takes the vector of real keys as input and creates a feasible solution for the problem: the decoder returns as outputs the number of wavelengths needed, the set of wavelengths that are effectively used, and the lightpaths – i.e., the routes – assigned to each demand. The decoder DECODER-SSLD will be described in Section 4.

We make use of the parametric uniform crossover strategy originally proposed in [43] for combining two parent solutions and producing a new one. According to this strategy, the solution generated by crossover inherits with a higher probability each of its keys from the best parent. BRKGAs, as the one developed here, do not make use of the standard mutation operator. Instead, the concept of mutants is used: new solutions (i.e., mutants) are introduced in the population at each generation, generated at random following the same strategy as in the initial population. They play the same role of the mutation operator in more standard genetic algorithm frameworks, diversifying the search and assisting the procedure to escape from local optima [8,9,30].

The $|D|$ real keys in each chromosome are randomly generated in the initial population. The population is divided in two subsets at each generation: TOP (formed by elite solutions) and $REST$. Subset TOP always contains the best (or elite) solutions. Subset $REST$ is also partitioned in two disjoint subsets: MID and BOT, with subset BOT containing the worst solutions. The solutions in TOP are copied to the population of the next generation, as shown in Figure 4. The solutions in BOT are replaced by new mutants, which are temporarily placed in the subset BOT of the population of the next generation. The rest of the solutions of the population of the next generation are obtained by crossover, with one parent always being randomly chosen from

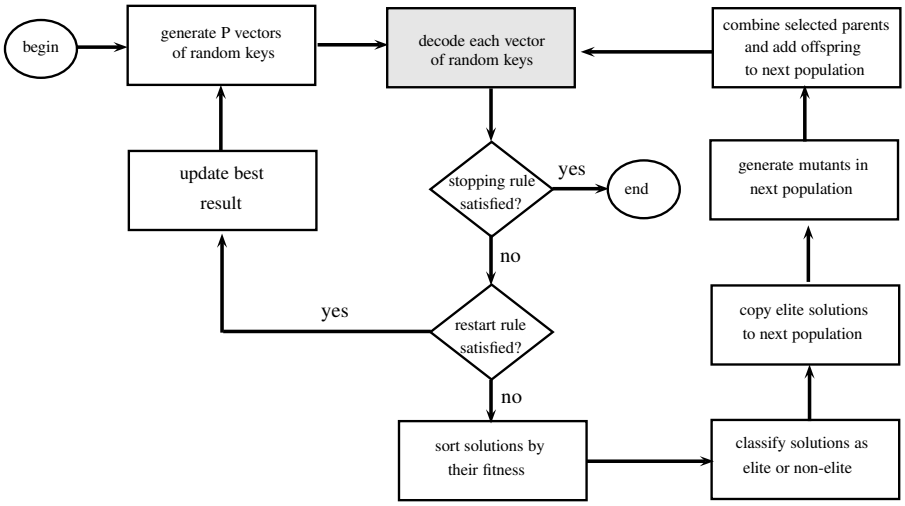


Fig. 5 BRKGA framework.

TOP and the other from *REST*. Therefore, $|MID| = |REST| - |BOT|$ solutions are created with this crossover strategy, which distinguishes a BRKGA from a simple RKGA, where both parents are randomly selected from the entire population. Since a parent solution can be chosen for crossover more than once in the same generation, elite solutions have a higher probability of passing their characteristics (i.e., their random keys) to the next generation.

4 Decoder and constructive heuristic

The implementation of a BRKGA for routing and wavelength assignment of sliding scheduled lightpath demands made use of the C++ library brkgaAPI developed by Toso and Resende [44], which is a framework for the implementation of biased random-key genetic algorithms.

The instantiation of the framework in Figure 5 requires exclusively the development of a class implementing the decoder for each specific problem, being the only problem-dependent part of the tool.

A BRKGA running in this framework requires some parameters [14]: (a) the population size $p = |TOP| + |REST|$; (b) the fraction *pe* of the population corresponding to the elite subset; (c) the fraction *pm* of the population corresponding to the mutant subset; (d) the probability *rhoe* that a solution obtained by crossover inherits its keys from the best parent; and (e) the number k of generations without improvement in the best solution until a new restart is performed.

The biased random-key genetic algorithm BRKGA-SSLD for solving RWA-SSLD makes use of the decoder DECODER-SSLD presented in Section 4.2. This decoder is built upon heuristic DP-RWA-SSLD described in the next section.

4.1 Heuristic DP_RWA_SSLD

Heuristic DP_RWA_SSLD to solve problem RWA-SSLD with sliding scheduled demands is a modification of algorithm DP_RWA_SLD to solve problem RWA-SSLD, already described in Section 2.1.1. It randomly sets the starting time of each demand d in the corresponding interval $[a_d, b_d - \tau_d]$. Algorithm DP_RWA_SLD is applied to the modified set of demands with randomly fixed starting times.

Algorithm 2 receives as inputs the graph $G = (V, E)$ representing the network and the set $D = D_{SSLD}$ of demands $d_1, \ldots, d_m$. As for the previous algorithm, its outputs are the number λ of wavelengths needed to route all demands, as well as the set W_d of wavelengths that may be used by demand d and the set of edges $Path_d$ in the route assigned to demand d, for every $d \in D$.

Heuristic DP_RWA_SSLD starts by initializing set D_{SLD} as empty in line 1. The loop in lines 2–9 progressively builds the set D_{SLD} from the demands in D_{SSLD}. For each demand $d \in D_{SSLD}$, a demand d' with the same source and destination nodes and the same number of required wavelengths is created in lines 3 to 5. The starting time $a_{d'}$ of demand d' is randomly selected in the interval $[a_d, b_d - \tau_d]$ in line 6. The finishing time of demand d' is set in line 7. Demand d' is inserted in set D_{SLD} in line 8. In line 10, demands $d_1, \ldots, d_m$ are renumbered in non-increasing order of their number of requested wavelengths. Ties are broken by first picking the demand with the largest shortest path from its source to its destination. Algorithm DP_RWA_SLD is applied in line 11 to the modified set of demands D_{SLD}. Line 12 returns the number of wavelengths needed, the wavelengths assigned to the demands, and the lightpaths used to route the demands.

Algorithm 2 DP_RWA_SSLD($G, D = D_{SSLD}$)

1: $D_{SLD} \leftarrow \emptyset$
2: **for all** $d \in D$ **do**
3: $\quad s_{d'} \leftarrow s_d$
4: $\quad t_{d'} \leftarrow t_d$
5: $\quad n_{d'} \leftarrow n_d$
6: $\quad$ Randomly select $a_{d'} \in [a_d, b_d - \tau_d]$
7: $\quad b_{d'} \leftarrow a_{d'} + \tau_d$
8: $\quad D_{SLD} \leftarrow D_{SLD} \cup \{d'\}$
9: **end for**
10: Renumber demands $d_1, \ldots, d_m$ in non-increasing order of their number of requested wavelenghts; in case of ties first pick the demand with the largest shortest path from its source to its destination.
11: $\lambda, (W_d, d = 1, \ldots, m), (Path_d, d = 1, \ldots, m) \leftarrow$ DP_RWA_SLD(G, D_{SLD})
12: **return** λ, $(W_d, d = 1, \ldots, m)$, $(Path_d, d = 1, \ldots, m)$

4.2 Decoder DECODER-SSLD

Every solution of the sliding scheduled lightpath demands problem (RWA-SSLD) is represented by a set of $m = |D_{SSLD}|$ random keys. Each key is a number in the interval $[0, 1)$, which is associated with a vertex of the graph. Each vector of random keys is decoded by an algorithm that receives the keys as inputs and constructs a feasible solution to RWA-SSLD: the decoder returns the number of wavelengths needed, the wavelengths assigned to the demands, and the lightpaths used to route the demands. The random keys are used to establish the starting times of the demands and interfere with the order in which they are renumbered by the decoder.

Decoder DECODER-SSLD to RWA-SSLD, whose pseudo-code is given by Algorithm 3, is based on the heuristic DP_RWA_SSLD. It receives the random keys $\alpha_d \in [0, 1), \forall d \in D_{SSLD}$, as additional parameters. The only adaptations with respect to the pseudo-code of the heuristic in Algorithm 2 appear in lines 6 and 10. By setting $a_{d'} = a_d + (b_d - \tau_d - a_d) \cdot \alpha_d$ in line 6, the starting time a'_d of demand d' can be set as small as a_d (in case the random key $\alpha_d = 0$) and very close to $b_d - \tau_d$ (in case α_d is close to one). In line 10, the random keys are used as a tie breaking criterion as the demands are renumbered.

Algorithm 3 DECODER-SSLD$(G, D = D_{SSLD}, \alpha)$

1: $D_{SLD} \leftarrow \emptyset$
2: **for all** $d \in D$ **do**
3: $\quad s_{d'} \leftarrow s_d$
4: $\quad t_{d'} \leftarrow t_d$
5: $\quad n_{d'} \leftarrow n_d$
6: $\quad a_{d'} \leftarrow a_d + (b_d - \tau_d - a_d) \cdot \alpha_d$
7: $\quad b_{d'} \leftarrow a_{d'} + \tau_d$
8: $\quad D_{SLD} \leftarrow D_{SLD} \cup \{d'\}$
9: **end for**
10: Renumber demands $d_1, \ldots, d_m$ in non-increasing order of their number of requested wavelenghts; in case of ties first pick the demand with the largest random key.
11: $\lambda, (W_d, d = 1, \ldots, m), (Path_d, d = 1, \ldots, m) \leftarrow$ DP_RWA_SLD(G, D_{SLD})
12: **return** λ, $(W_d, d = 1, \ldots, m)$, $(Path_d, d = 1, \ldots, m)$

5 Numerical experiments

In this section, we evaluate the performance of the biased random-key genetic algorithm for RWA-SSLD, based on the decoder DECODER-SSLD described in Section 4.2.

The algorithm was implemented in C++ with the GNU GCC compiler C/C++ version 5.4.0. The experiments were performed on a i7-6500U processor with a 2.50 GHz clock and 8 GB of RAM running the operating system Linux Ubuntu 16.04 LTS with parallel processing features disabled.

5.1 Instance generation

Data from different randomly generated test problems that have been used to evaluate algorithms previously developed for variants of RWA-SLD and RWA-SSLD [4,18,20,22,24,41,45,46] are not available from the literature.

Each problem instance is defined by a network topology (the graph G) and by a set of sliding scheduled demands (the set D_{SSLD}). The test instances considered in this work use the same graphs provided by Noronha, Resende and Ribeiro [29] and described in Table 2. The demands have been generated by the original source code developed by Kuri et al. [22], which was also used in [41]. The code made available by the authors has a parameter $C \in [0, 1]$ that establishes the time correlation between the demands. A time correlation close to 0 means that the demands are weakly time correlated, while a time correlation close to 1 means that the demands are strongly time correlated. Smaller time correlations produce sets of demands with less overlapping.

Table 2 Network topologies used in the instances generated for the numerical experiments.

Topology	Nodes	Edges
ATT	90	274
Brazil	27	140
EON	20	78
Finland	31	102
NSF	14	42

Each demand d generated for RWA-SLD by the code in [22] is transformed into a SSLD demand d' by the creation of a sliding window from its starting and finishing times a_d and b_d, respectively. The sliding window for the corresponding SSLD demand d' is $[a_{d'}, b_{d'}]$, with $a_{d'} = a_d - \gamma_1(b_d - a_d)$ and $b_{d'} = b_d + \gamma_2(b_d - a_d)$, with γ_1 and γ_2 randomly selected in the interval $[0, B]$.

The computational experiments reported in the next section have been performed on 60 randomly generated test instances: for each of the five network topologies described in Table 2, 12 instances have been generated with the number of demands $m \in [100, 10000]$, the time correlation $C \in [0.600, 0.994]$, and $B \in \{2, 4, 10\}$.

5.2 Parameter tuning

The best parameter settings for algorithm BRKGA-SSLD have been determined using the automatic tuning tool IRACE [3,25,32]. We investigated the best settings for the following parameters: p (population size), pe (fraction of the population corresponding to the elite subset), pm (fraction of the population corresponding to the mutant subset), and $rhoe$ (probability that a solution generated by crossover inherits its keys from the best parent).

Table 3 Test instances (15) used in the tuning experiment with IRACE.

Instance	Demands
ATT100.C990.B2	100
ATT1000.C900.B2	1000
ATT8000.C950.B10	8000
Brazil100.C980.B2	100
Brazil2000.C900.B10	2000
Brazil4000.C990.B4	4000
EON500.C900.B2	500
EON1000.C950.B4	1000
EON10000.C990.B10	10000
Finland100.C900.B2	100
Finland1000.C600.B2	1000
Finland3000.C994.B10	3000
NSF200.C900.B4	200
NSF2000.C950.B2	2000
NSF8000.C990.B10	8000

The IRACE tuning experiment automatically performed 1000 runs [7,27] of BRKGA-SSLD on 15 additional problem instances generated as described in Section 5.1 and listed in Table 3. Each of these runs was limited to one hour and we tested the value ranges suggested by Gonçalves and Resende [13]. The value ranges used by IRACE and the best values obtained for each of the parameters are reported in Table 4.

Table 4 Value ranges used by IRACE and best parameter values.

Parameter	Value ranges	BRKGA-SSLD
p	50, 51, ..., 100	83
pe	0.10, 0.11, ..., 0.25	0.17
pm	0.10, 0.11, ..., 0.30	0.20
$rhoe$	0.50, 0.51, ..., 0.80	0.61

5.3 Numerical results

We first compared the proposed algorithm BRKGA-SSLD with a straightforward multistart heuristic for RWA-SSLD implemented by Algorithm 4, which receives as inputs the graph $G = (V, E)$ representing the network and the set $D = D_{SSLD}$ of demands $d_1, \ldots, d_m$. Once again, its outputs are the number λ of wavelengths needed to route all demands, as well as the set W_d of wavelengths that may be used by demand d and the set of edges $Path_d$ in the route assigned to demand d, for every $d \in D$. The multistart heuristic MDP_RWA_SSLD repeatedly applies the randomized heuristic DP_RWA_SSLD, until some stopping criterion is reached. The number of consecutive iterations without improvement and the number of wavelengths needed to route all demands are initialized in lines 1 and 2, respectively. The

loop in lines 3–15 implements the stopping criterion. Line 4 performs a call to the randomized heuristic DP_RWA_SSLD. If the number λ' of wavelengths in the new solution does not improve the number λ of wavelengths in the incumbent, then the number of iterations without improvement is updated in line 6. Otherwise, the number of iterations without improvement is reset to zero in line 8 and the incumbent is updated in lines 9–13. Line 16 returns the number of wavelengths needed, the wavelengths assigned to the demands, and the lightpaths used to route the demands in the best solution found.

Algorithm 4 MDP_RWA_SSLD($G, D = D_{SSLD}$)

```
1: niter ← 0
2: λ ← ∞
3: while stopping criterion is not reached do
4:    λ', (W'_d, d = 1,...,m), (Path'_d, d = 1,...,m) ← DP_RWA_SSLD(G, D)
5:    if λ' ≥ λ then
6:       niter ← niter + 1
7:    else
8:       niter ← 0
9:       λ ← λ'
10:      for all d ∈ D do
11:         W_d ← W'_d
12:         Path_d ← Path'_d
13:      end for
14:   end if
15: end while
16: return λ, (W_d, d = 1,...,m), (Path_d, d = 1,...,m)
```

We considered the 60 test instances generated in Section 5.1 for the comparative evaluation of heuristic BRKGA-SSLD proposed in this work with the MDP_RWA_SSLD multistart heuristic. We observe that these test instances do not contain any of the instances considered in the tuning experiment. Ten independent runs using different seeds were performed for each algorithm. Algorithm MDP_RWA_SSLD was made to stop after 1000 iterations without improvement in the best known solution. For each test instance, we used the average time taken by algorithm MDP_RWA_SSLD as the stopping criterion for BRKGA-SSLD. Therefore, the same stopping criterion is applied to both heuristics.

For each instance, Tables 5 and 6 present the best and average solution values over ten runs of each algorithm. The last column in each of these tables shows the computation time for MDP_RWA_SSLD, which was also taken as the stopping criterion for BRKGA-SSLD. Cells in boldface show the algorithms that obtained the best results for each instance. These results show that BRKGA-SSLD obtained systematically better solutions than MDP_RWA_SSLD. While MDP_RWA_SSLD found a solution for only one instance (EON8000.C950.B10) that was not matched by BRKGA-SSLD, the latter found unmatched solutions by MDP_RWA_SSLD for 29 out of the 60 test problems. Similarly, MDP_RWA_SSLD found the best average solution

Table 5 Results for algorithms BRKGA-SSLD and MDP_RWA_SSLD - Part I.

		MDP_RWA_SSLD		BRKGA-SSLD		Running
Instance	Demands	Best	Average	Best	Average	time (s)
ATT100.C800.B10	100	**12**	**12.00**	**12**	**12.00**	1.507
ATT100.C900.B2	100	**14**	**14.00**	**14**	**14.00**	1.507
ATT500.C800.B2	500	13	13.10	**12**	**12.80**	12.296
ATT500.C900.B4	500	14	14.00	**13**	**13.90**	14.248
ATT500.C950.B10	500	**15**	15.80	**15**	**15.10**	11.582
ATT1000.C800.B4	1000	14	14.10	**13**	**13.40**	27.615
ATT1000.C990.B10	1000	24	24.50	**23**	**23.30**	32.726
ATT2000.C500.B2	2000	**15**	**15.00**	**15**	**15.00**	57.273
ATT2000.C900.B4	2000	19	19.00	**18**	**18.90**	56.808
ATT4000.C900.B4	4000	**19**	**19.00**	**19**	**19.00**	177.383
ATT8000.C990.B10	8000	**30**	30.70	**30**	**30.00**	839.191
ATT10000.C994.B10	10000	**30**	30.70	**30**	**30.20**	1189.970
Brazil200.C980.B2	200	**10**	**10.00**	**10**	**10.00**	1.390
Brazil400.C980.B2	400	**13**	**13.00**	**13**	**13.00**	3.289
Brazil600.C980.B2	600	**12**	**12.10**	**12**	12.40	5.950
Brazil800.C980.B2	800	17	17.00	**14**	**16.50**	8.090
Brazil1000.C950.B10	1000	11	11.60	**10**	**11.00**	15.222
Brazil2000.C950.B10	2000	12	12.80	**11**	**11.90**	46.812
Brazil3000.C950.B10	3000	**15**	15.70	**15**	**15.00**	107.099
Brazil4000.C950.B10	4000	13	13.00	**12**	**12.50**	146.872
Brazil5000.C950.B10	5000	**18**	**18.00**	**18**	**18.00**	208.167
Brazil6000.C950.B10	6000	**15**	**15.00**	**15**	**15.00**	314.619
Brazil7000.C950.B10	7000	17	17.00	**16**	**16.90**	405.849
Brazil8000.C950.B10	8000	20	20.00	**16**	**16.10**	546.530
EON200.C900.B10	200	**10**	**10.00**	**10**	**10.00**	1.010
EON200.C990.B2	200	16	16.70	**15**	**15.40**	1.659
EON1000.C950.B10	1000	**13**	**13.00**	**13**	**13.00**	10.470
EON1000.C990.B2	1000	24	24.00	**22**	**23.30**	12.980
EON4000.C950.B10	4000	**17**	17.10	**17**	**17.00**	169.986
EON5000.C994.B4	5000	22	22.40	**21**	**21.70**	254.200
EON6000.C990.B10	6000	**24**	25.10	**24**	**24.40**	435.820
EON7000.C990.B10	7000	24	24.40	**23**	**23.50**	486.826
EON8000.C950.B10	8000	**19**	**19.20**	20	20.00	669.642
EON8000.C994.B4	8000	**24**	25.00	**24**	**24.70**	623.265
EON9000.C990.B10	9000	**25**	25.60	**25**	**25.10**	924.462
EON10000.C994.B10	10000	**25**	26.10	**25**	**25.70**	997.888
Finland100.C950.B4	100	**10**	10.30	**10**	**10.00**	0.915
Finland200.C950.B4	200	**17**	**17.00**	**17**	**17.00**	1.365
Finland300.C950.B4	300	14	14.00	**13**	**13.90**	3.278
Finland400.C950.B4	400	14	14.20	**13**	**13.00**	4.150
Finland1000.C950.B4	1000	16	16.00	**15**	**15.90**	12.678
Finland1000.C990.B2	1000	18	18.60	**17**	**17.20**	16.293

values for only two instances (Brazil600.C980.A2 and EON8000.C950.A10), while MDP_RWA_SSLD found the best averages for 43 instances. Table 7 summarizes the following results from the above tables:

– *#Best*: number of test instances where some algorithm obtained the best value (the higher this value, the better is the corresponding algorithm).

Table 6 Results for algorithms BRKGA-SSLD and MDP_RWA_SSLD - Part II.

		MDP_RWA_SSLD		BRKGA-SSLD		Running
Instance	Demands	Best	Average	Best	Average	time (s)
Finland2000.C950.B4	2000	18	18.00	**17**	**17.30**	38.067
Finland2000.C990.B2	2000	22	22.00	**21**	**21.80**	38.822
Finland3000.C990.B2	3000	**23**	23.70	**23**	**23.30**	88.459
Finland4000.C990.B2	4000	24	24.00	**23**	**23.40**	167.492
Finland8000.C994.B10	8000	25	25.00	**24**	**24.20**	542.394
Finland10000.C994.B10	10000	**25**	**25.90**	**25**	**25.90**	1041.525
NSF200.C950.B2	200	15	15.50	**14**	**14.40**	1.043
NSF200.C980.B10	200	**16**	**16.00**	**16**	**16.00**	1.116
NSF300.C900.B4	300	**17**	**17.00**	**17**	**17.00**	1.271
NSF300.C990.B2	300	24	24.10	**22**	**22.00**	1.915
NSF500.C900.B4	500	**17**	**17.00**	**17**	**17.00**	2.740
NSF800.C990.B4	800	25	25.70	**24**	**24.80**	8.138
NSF1000.C880.B10	1000	20	20.00	**16**	**16.90**	8.460
NSF2000.C920.B2	2000	**17**	17.70	**17**	**17.50**	37.105
NSF4000.C990.B2	4000	**30**	30.80	**30**	**30.00**	142.126
NSF6000.C980.B4	6000	28	28.00	**26**	**26.40**	321.744
NSF8000.C994.B4	8000	29	29.50	**28**	**28.80**	567.039
NSF10000.C994.B10	10000	**31**	31.90	**31**	**31.50**	819.027

Table 7 Comparative results for the multistart heuristic and the biased random-key genetic algorithm.

	MDP_RWA_SSLD	BRKGA-SSLD
#Best	31	**59**
#BestM	17	**58**
SumBest	202	**374**
Avgdevrun (%)	3.83	**1.49**
Avgdev (%)	4.04	**0.09**
AvgdevM (%)	3.10	**0.11**

- *#BestM*: number of test instances where some algorithm obtained the best average value (the higher this value, the better is the corresponding algorithm).
- *SumBest*: number of runs where some algorithm obtained the best value (the higher this value, the better is the corresponding algorithm).
- *Avgdevrun*: average relative deviation between the value obtained by some algorithm over all runs of some instance and the best value obtained (for this instance) considering all tested algorithms (the smaller this value, the better is the corresponding algorithm).
- *Avgdev*: average relative deviation between the best value obtained by some algorithm for some instance and the best value obtained (for this instance) considering all tested algorithms (the smaller this value, the better is the corresponding algorithm).
- *AvgdevM*: average relative deviation between the average value obtained by some heuristic for some instance and the best average value obtained (for this instance) considering all tested algorithms (the smaller this value, the better is the corresponding algorithm).

The results summarized in Table 7 make the case for BRKGA-SSLD. They show that BRKGA-SSLD consistently obtained the best results for all criteria considered, finding 59 best solution values and 58 best average values out of the 60 instances. BRKGA-SSLD was also the heuristic that found the solutions leading to the smallest values for the results reporting average relative deviations from the best values.

In the following, we compare the runtime distributions (also known as time-to-target plots or ttt-plots) of heuristics BRKGA-SSLD and MDP_RWA_SSLD. These 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. Time-to-target plots have also been claimed in [15] as a very effective tool to characterize the behavior of running times of stochastic local search algorithms. In the experiment reported in the sequel, the two heuristics are made to stop whenever a solution with cost smaller than or equal to a given target is found. The target is set as the cost of the best known solution for the instance. 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 $i = 1, \ldots, 200$. The more to the left is a plot, the better is the corresponding algorithm.

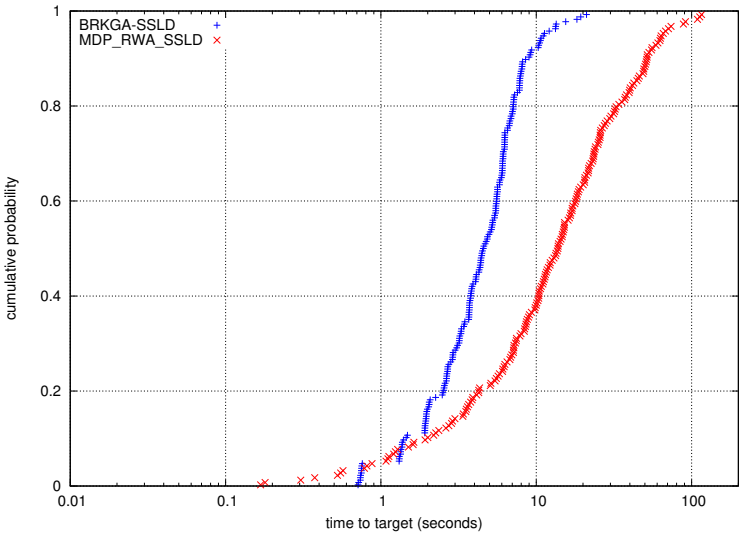


Fig. 6 Runtime distributions for instance ATT500.C950.B10 with target set at 15.

Time-to-target plots for instance ATT500.C950.B10 are displayed in Figure 6, with the target set to 15 (best solution value obtained by all heuristics

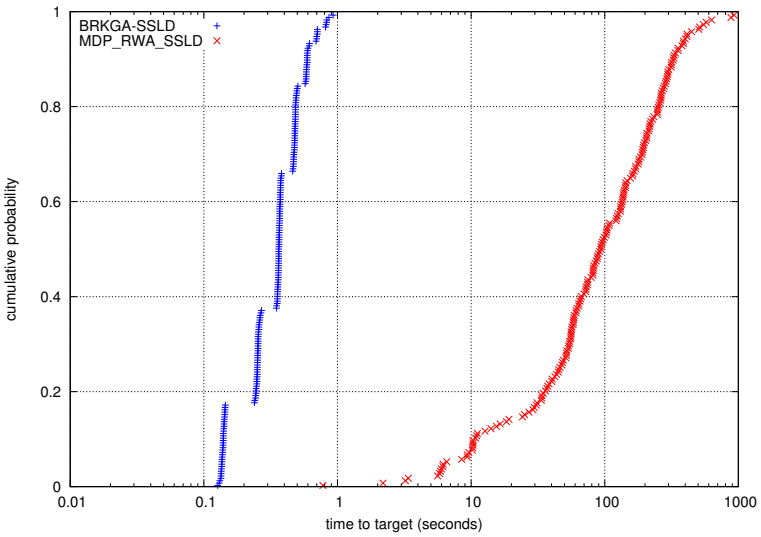


Fig. 7 Runtime distributions for instance NSF300.C990.B2 with target set at 23.

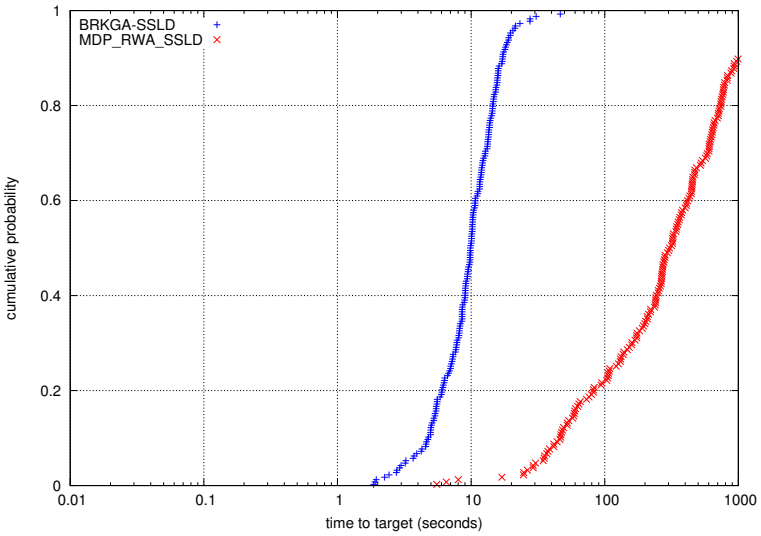


Fig. 8 Runtime distributions for instance NSF800.C990.B4 with target set at 24.

in Table 5). Plots for instance NSF300.C990.B2 appear in Figure 7, with the target set to 23 (average between the solution values obtained by heuristics BRKGA-SSLD and MDP_RWA_SSLD in Table 6). The ttt-plots for instance NSF800.C990.B4 displayed in Figure 8, with the target set to 24 (best value found by BRKGA-SSLD in Table 6) show that MDP_RWA_SSLD failed to reach the target within the time limit in 19 runs. The ttt-plots show that,

given the same running time, BRKGA-SSLD is capable to find with higher probability than MDP_RWA_SSLD solutions that are as good as the targets.

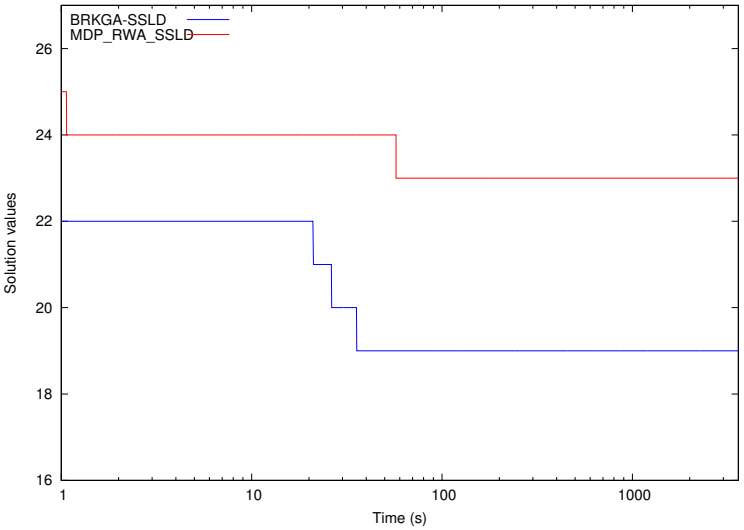


Fig. 9 Evolution of the best solution value obtained by BRKGA-SSLD and MDP_RWA_SSLD along the 3600 first seconds of running time for instance NSF300.C990.B2: best value found by BRKGA-SSLD is 19.

Figures 9 and 10 show how the best solutions encountered by BRKGA-SSLD and the multistart heuristic evolve along the first 3600 seconds of running time for each of the instances NSF300.C990.B2 and NSF800.C990.B4, respectively. They show that BRKGA-SSLD obtains better solutions faster than the multistart algorithm. It is capable to continuously evolve the solution population and to improve the best value. The best solution found by BRKGA-SSLD is better than that obtained by MDP_RWA_SSLD anytime along these runs.

Restart strategies are very effective in order to reduce the running times to reach target values. We hybridized heuristic BRKGA-SSLD with restart(κ) strategy first proposed in [35,37]. In this context, the population is completely renewed after that κ generations have been performed without improvement in the best solution obtained, with κ being a new parameter. The performance of strategy restart(κ) was evaluated for $\kappa = 100$, 200, and 500.

Figure 11 displays runtime distributions for all variants of BRKGA-SSLD, with and without restarts, on instance NSF300.C990.B2. For each strategy restart(κ), 200 runs have been performed. The running time of each run was limited to 1000 seconds and the target value was set to 20. The figure shows that BRKGA-SSLD without restarts failed to reach the target in 21 runs. Figure 12 displays runtime distributions for all variants of BRKGA-SSLD, with and without restarts, on instance NSF800.C990.B4. The running time of

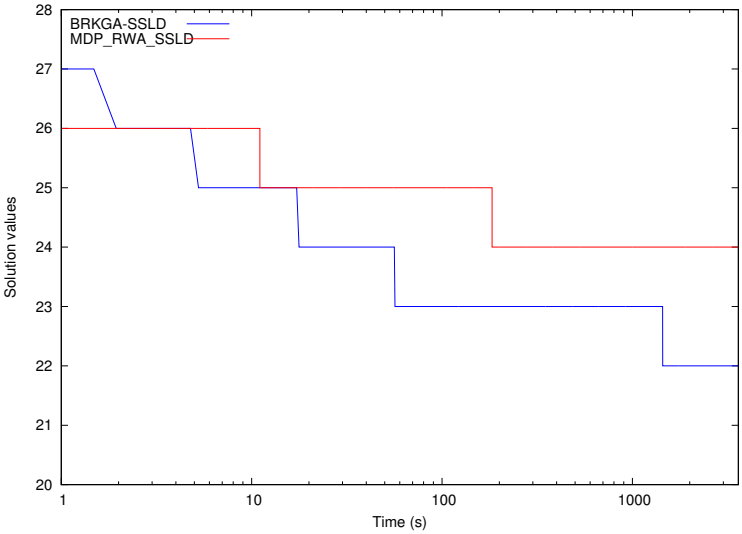


Fig. 10 Evolution of the best solution value obtained by BRKGA-SSLD and MDP_RWA_SSLD along the 3600 first seconds of running time for instance NSF800.C990.B4: best value found by BRKGA-SSLD is 22.

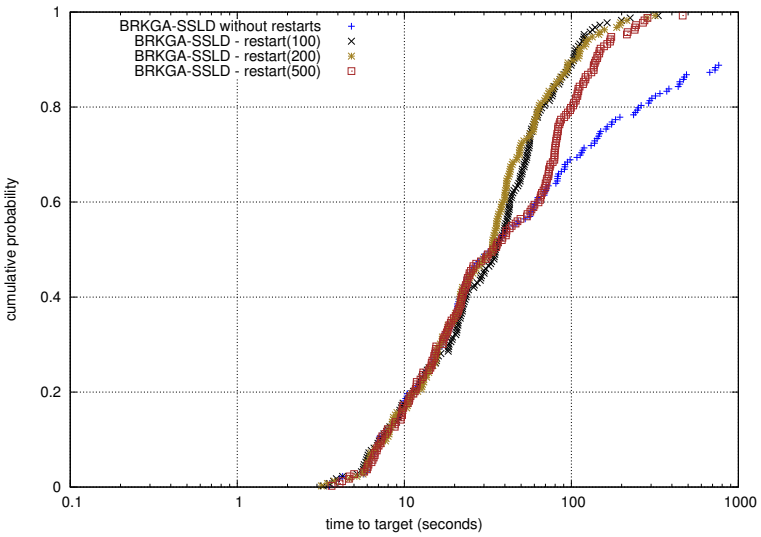


Fig. 11 Runtime distributions for heuristic BRKGA-SSLD with restart(κ) strategies on instance NSF300.C990.B2 with target set at 20.

each run was limited to 10000 seconds and the target value was set to 22. The figure shows that BRKGA-SSLD without restarts failed to reach the target in 18 runs. For these two instances, strategies restart(100) and restart(200) presented the best results, corresponding to the leftmost plots.

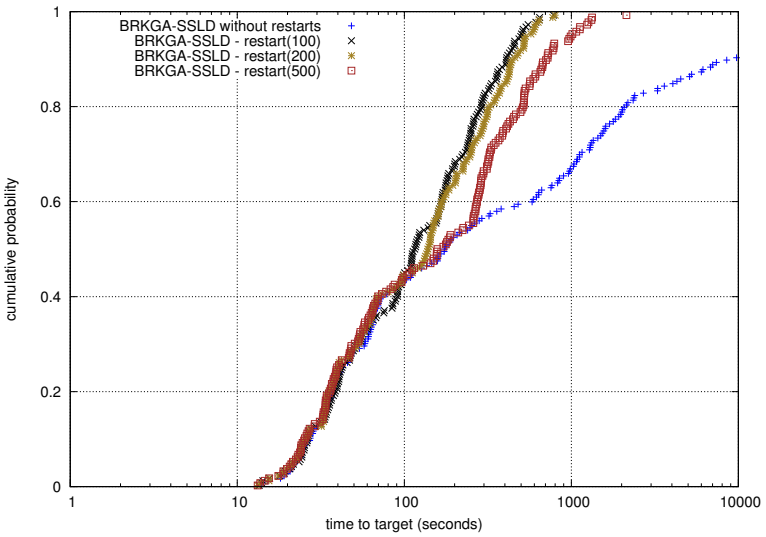


Fig. 12 Runtime distributions for heuristic BRKGA-SSLD with restart(κ) strategies on instance NSF800.C990.B4 with target set at 22.

Table 8 Restart strategies on instance NSF800.C990.B4: compared to the strategy without restarts, strategies restart(100) and restart(200) reduced the average running time to less than 11% of the value without restarts.

	Average running times in quartile (seconds)				
Strategy	1st	2nd	3rd	4th	average
BRKGA-SSLD without restarts	29.45	82.88	713.29	5984.37	1702.50
BRKGA-SSLD with restart(100)	29.63	77.29	<u>178.36</u>	<u>391.49</u>	<u>169.19</u>
BRKGA-SSLD with restart(200)	28.51	<u>76.61</u>	194.44	442.77	185.58
BRKGA-SSLD with restart(500)	<u>28.21</u>	77.90	278.58	730.27	278.74

Resende and Ribeiro [35,37] observed that the effect of restart strategies is mainly noticed in the longest runs. Table 8 illustrates the behavior of the restart strategies on instance NSF800.C990.B4, considering 200 runs of each variant, the target set at 22, and a time limit of 10000 seconds. We focus on the columns of this table corresponding to the third and fourth quartiles, whose entries correspond to running times in the heavy tails of the runtime distributions. The restart strategies strongly affect mostly these two quartiles, which is a desirable characteristic. Compared to the strategy without restarts, strategies restart(100) and restart(200) were both able to reduce the average running time to less than 11% of the original value. The best results for each quartile are underlined. Algorithm BRKGA-SSLD with strategy restart(100) outperformed the other variants tested, with the smallest average running times. In the fourth quartile, BRKGA-SSLD without restarts failed to reach the target in 18 runs.

The results of these experiments show that BRKGA-SSLD together with restart(100) or restart(200) strategies performed the best over all tested vari-

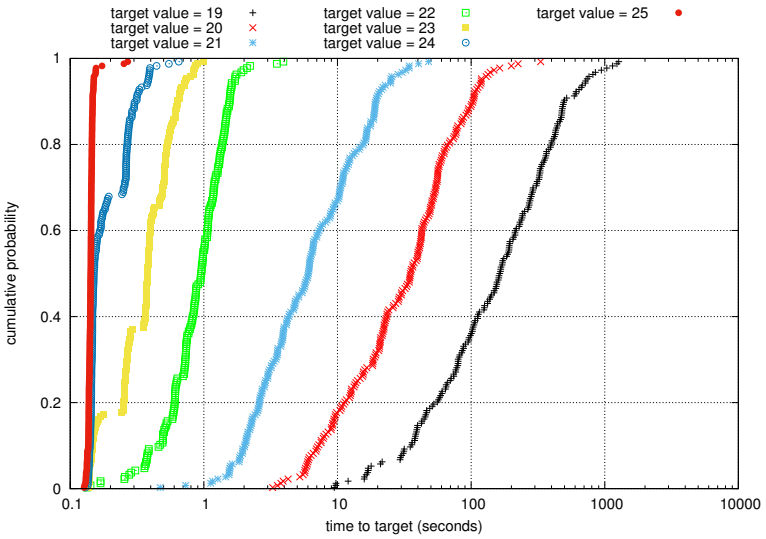


Fig. 13 Runtime distributions for heuristic BRKGA-SSLD with restart(100) strategy on instance NS300.C990.B2 as the target value ranges from 19 to 25.

ants, illustrating that restart strategies are capable to reduce the execution times of long runs of metaheuristics for optimization problems, although their use has been systematically neglected in the literature.

To conclude the experiments in this section, Figure 13 displays ttt-plots for BRKGA-SSLD with strategy restart(100) on instance NSF300.C990.B2 with the running time limited to 10000 seconds as the target ranges from 19 to 25. Figure 14 shows the average running time (in seconds) over 200 runs for each target value. We observe that, as the target decreases and the problem gets harder, the search strategy takes longer to reach the target.

5.4 Comparing BRKGA-SSLD with MCDF-CR

In the final experiment, we compare the proposed heuristic BRKGA-SSLD with the MCDF-CR algorithm from [24]. Comparisons with other algorithms are not possible, either because they handle different problem variants or because there is not enough information available about the experiments reported in the literature.

Since the source or executable codes of algorithm MCDF-CR, as well as the test problems for the numerical experiments reported in [24] were not available from the authors, we randomly generated 100 test instances, divided into 10 groups, with the number of demands m ranging from 10 to 100, using the same parameters described in [24].

Always following Liu et al. [24], and in order to reproduce as closely as possible the same conditions tested in their experiments, we used the same US

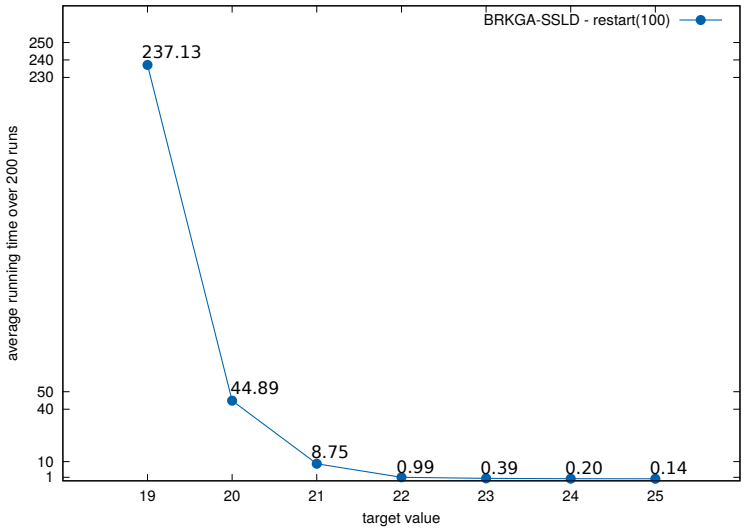


Fig. 14 Average running times (seconds) for heuristic BRKGA-SSLD with restart(100) strategy on instance NSF300.C990.B2 as the target ranges from 19 to 25.

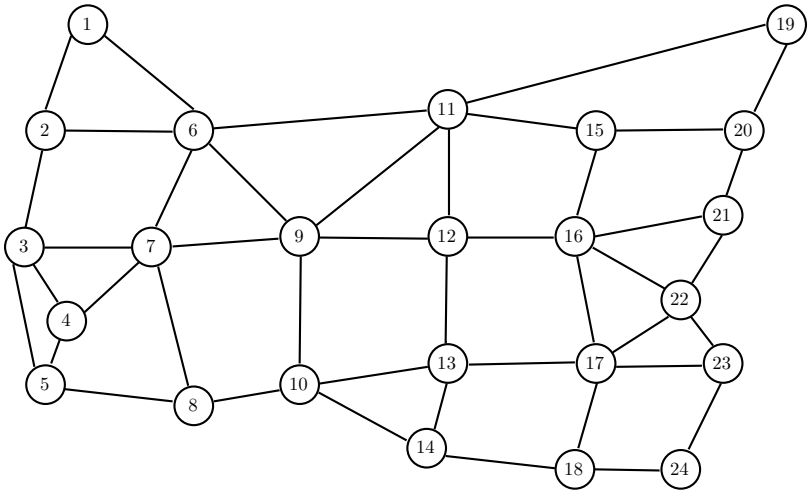


Fig. 15 US nationwide topology with 24 nodes and 43 edges.

nationwide topology with 24 nodes and 43 edges [24] depicted in Figure 15. We considered a 24-hour period sliced into 48 time slots of 30 minutes each. The source node s_d and the destination node t_d of each demand d were randomly selected. The starting and finishing times of the sliding window of this demand were defined as $a_d = \min(t_1, t_2)$ and $b_d = \max(t_1, t_2)$, respectively, with t_1 and t_2 randomly generated in the interval $[0, 48]$, while its duration τ_d was randomly generated in the interval $[0.5 \times (b_d - a_d), (b_d - a_d)]$.

Table 9 Average number of wavelengths over ten instances with the same number of demands for algorithms MCDF-CR [24] and BRKGA-SSLD (running for one, ten, and 100 generations).

		BRKGA-SSLD		
Demands	MCDF-CR	1 generation	10 generations	100 generations
10	1.71	1.20	1.20	1.20
20	2.64	1.52	1.42	1.40
30	3.43	1.92	1.71	1.63
40	4.13	2.20	2.06	2.00
50	5.04	2.31	2.25	2.09
60	5.67	2.91	2.78	2.67
70	6.40	3.16	3.06	2.91
80	7.09	3.39	3.18	3.10
90	7.34	3.87	3.63	3.39
100	8.24	4.62	4.18	4.10

In order to make the heuristic BRKGA-SSLD comparable with MCDF-CR, the right-hand side of the inequality in the test in line 15 of Algorithm 1 has to be changed from $\max\{diam_G, \sqrt{|E|}\}$ to $c_{s_d,t_d}(G) + \delta$, meaning that the length of any lightpath cannot be longer than the shortest path between its extremities by more than δ hops. We used $\delta = 1$, to reproduce the experiment in the work of Liu et al. [24].

Table 9 displays the average number of wavelengths calculated by algorithms MCDF-CR and BRKGA-SSLD. The number of wavelengths shown for MCDF-CR are average results obtained over 50 runs of the algorithm in [24]. Since the original numerical results were not available, we needed to extract data from the plot in Figure 8 of [24]. We used WebPlotDigitizer [11,38], a web-based plot digitizing tool, for extracting data from a variety of plots, including the coordinates of interrupted time-series data. The average results for BRKGA-SSLD are obtained over 100 runs of the heuristic for each problem size, corresponding to ten runs of each of the ten instances with the same number of demands. Results are given for BRKGA-SSLD running for one, ten, and 100 generations.

The results obtained by BRKGA-SSLD and MCDF-CR displayed in Table 9 are reproduced in Figure 16. This figure shows that BRKGA-SSLD clearly outperforms MCDF-CR, finding solutions that make use of roughly 50% of the number of wavelengths determined by MCDF-CR. In addition, we observe that the performance ratio between BRKGA-SSLD and MCDF-CR increases as the number of demands gets larger.

6 Concluding remarks

We developed a biased random-key genetic algorithm for approximately solving the problem of routing and wavelength assignment of sliding scheduled lightpath demands (RWA-SSLD). In this problem variant, each demand is characterized not only by a source and a destination, but also by a duration and a time window in which it has to be met.

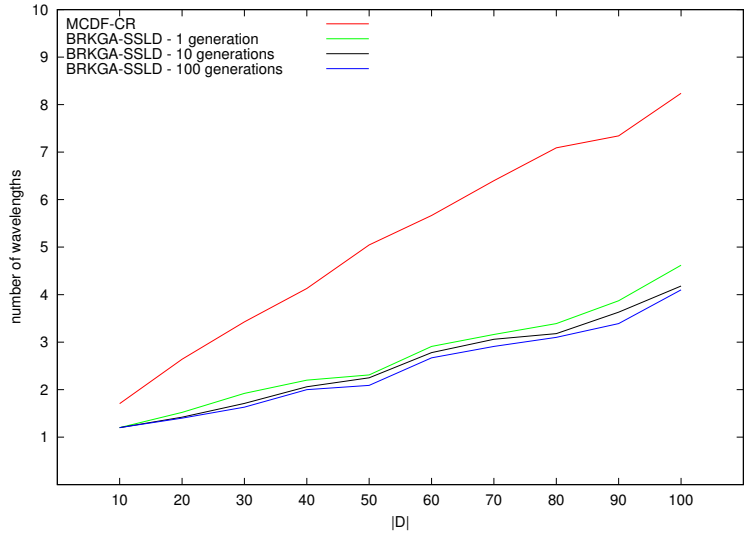


Fig. 16 Comparative results for algorithms BRKGA-SSLD and MCDF-CR [24].

The decoder for heuristic BRKGA-SSLD is based on a randomized extension of the adaptive greedy heuristic DP_RWA_SLD originally proposed by Skorin-Kapov [41] for solving a simpler variant of the problem (RWA-SLD). The latter is based on the adaptive greedy heuristic of Manohar et al. [26] for the original min-RWA problem without time constraints.

Although restart strategies are known to be capable to reduce the execution times of long runs of metaheuristics for optimization problems [35,37], their use has been systematically neglected in the literature. The numerical experiments also showed that the restart strategies gave a significant contribution to improve the robustness and the efficiency of the algorithm, since the BRKGA-SSLD algorithm combined with a restart strategy outperformed the original heuristic.

The results obtained by the biased random-key genetic algorithm are also compared with those obtained by Liu et al. [24], in a experiment reproducing as closely as possible the same conditions tested in their experiments. The new heuristic outperformed the algorithm in the literature, finding solutions that use roughly 50% of the number of wavelengths determined by the latter.

The numerical results supporting these findings illustrate the efficiency and the effectiveness of the BRKGA approach for solving the problem of routing and wavelength assignment of sliding scheduled demands, as well as other routing and scheduling problems [36]. In addition, we have made available through Mendeley [33] a complete data set of benchmark instances, with all the input data used in the numerical experiments performed in this work.

Acknowledgements The authors are grateful to J. Kuri for sending the Perl script for generating the test data. Work of Celso C. Ribeiro was partially supported by CNPq research

grant 303958/2015-4 and by FAPERJ research grant E-26/201.198/2014, and concluded during a visit to Laboratoire d'Informatique de Modélisation et d'Optimisation des Systèmes of Université Clermont Auvergne, France. This work was also partially sponsored by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Finance Code 001.

References

1. R. M. Aiex, M. G. C. Resende, and C. C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8:343–373, 2002.
2. R. M. Aiex, M. G. C. Resende, and C. C. Ribeiro. TTTPLOTS: A Perl program to create time-to-target plots. *Optimization Letters*, 1:355–366, 2007.
3. P. Alfaro-Fernández, R. Ruiz, F. Pagnozzi, and T. Stützle. Exploring automatic algorithm design for the hybrid flowshop problem. In *12th Metaheuristics International Conference*, pages 201–203, Barcelona, 2017.
4. D. Andrei, H. H. Yen, M. Tornatore, C. U. Martel, and B. Mukherjee. Integrated provisioning of sliding scheduled services over WDM optical networks. *IEEE/OSA Journal of Optical Communications and Networking*, 1:94–105, 2009.
5. D. Banerjee and B. Mukherjee. A practical approach for routing and wavelength assignment in large wavelength-routed optical networks. *IEEE Journal on Selected Areas in Communications*, 14:903–908, 1996.
6. J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6:154–160, 1994.
7. S. Bouamama and C. Blum. A population-based iterated greedy algorithm for the knapsack problem with setup. In *12th Metaheuristics International Conference*, pages 558–565, Barcelona, 2017.
8. 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. *International Transactions in Operational Research*, 22:823–839, 2015.
9. 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. *International Transactions in Operational Research*, 24:1061–1077, 2016.
10. J. S. Brandão, T. F. Noronha, and C. C. Ribeiro. A biased random-key genetic algorithm to maximize the number of accepted lightpaths in WDM optical networks. *Journal of Global Optimization*, 65:813–835, 2016.
11. D. Drevon, S. R. Fursa, and A. L. Malcolm. Intercoder reliability and validity of WebPlotDigitizer in extracting graphed data. *Behavior Modification*, 41:323–339, 2017.
12. T. Erlebach and K. Jansen. The complexity of path coloring and call scheduling. *Theoretical Computer Science*, 255:33–50, 2001.
13. J. F. Gonçalves and M. G. C. Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17:487–525, 2011.
14. 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.
15. H. H. Hoos and T. Stützle. Evaluation of Las Vegas algorithms - Pitfalls and remedies. In G. Cooper and S. Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 238–245, Madison, 1998.
16. E. Hyytia and J. Virtamo. Wavelength assignment and routing in WDM networks. In *Fourteenth Nordic Teletraffic Seminar*, pages 31–40, Copenhagen, 1998.
17. R. R. Iraschko, M. H. MacGregor, and W. D. Grover. Optimal capacity placement for path restoration in STM or ATM mesh-survivable networks. *IEEE/ACM Transactions on Networking*, 6:325–336, 1998.
18. A. Jaekel and Y. Chen. Demand allocation without wavelength conversion under a sliding scheduled traffic model. In *Fourth International Conference on Broadband Communications, Networks and Systems*, pages 495–503, Raleigh, 2007.
19. A. Jaekel and Y. Chen. Resource provisioning for survivable WDM networks under a sliding scheduled traffic model. *Optical Switching and Networking*, 6:44–54, 2009.

20. A. Jaekel, J. Pare, Y. Chen, A. Shaabana, and F. Luo. Traffic grooming of scheduled demands for minimizing energy consumption. *Photonic Network Communications*, 29:151–163, 2015.
21. J. M. Kleinberg. *Approximation algorithms for disjoint paths problems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, 1996.
22. J. Kuri, N. Puech, M. Gagnaire, E. Dotaro, and R. Douville. Routing and wavelength assignment of scheduled lightpath demands. *IEEE Journal on Selected Areas in Communications*, 21:1231–1240, 2003.
23. G. Li and R. Simha. The partition coloring problem and its application to wavelength routing and assignment. In *Proceedings of the First Workshop on Optical Networks*, Dallas, 2000.
24. Z. Liu, W. Guo, Q. Shi, W. Hu, and M. Xia. Sliding scheduled lightpath provisioning by mixed partition coloring in WDM optical networks. *Optical Switching and Networking*, 10:44–53, 2013.
25. M. López-Ibánez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The IRACE package: Iterated race for automatic algorithm configuration, 2011. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium.
26. P. Manohar, D. Manjunath, and R. K. Shevgaonkar. Routing and wavelength assignment in optical networks from edge disjoint path algorithms. *IEEE Communications Letters*, 6:211–213, 2002.
27. J. Maschler, T. Hackl, M. Riedler, and G. R. Raidl. Enhanced iterated greedy metaheuristic for the particle therapy patient scheduling problem. In *12th Metaheuristics International Conference*, pages 118–127, Barcelona, 2017.
28. T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro. Efficient implementations of heuristics for routing and wavelength assignment. In C. C. McGeoch, editor, *Experimental Algorithms*, volume 5038 of *Lecture Notes in Computer Science*, pages 169–180. Springer, Berlin, 2008.
29. T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro. Instances for the routing and wavelength assignment problem. http://www2.ic.uff.br/~celso/grupo/rwa.html, 2010. Last accessed October 13, 2018.
30. T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro. A biased random-key genetic algorithm for routing and wavelength assignment. *Journal of Global Optimization*, 50:503–518, 2011.
31. T. F. Noronha and C. C. Ribeiro. Routing and wavelength assignment by partition colouring. *European Journal of Operational Research*, 171:797–810, 2006.
32. 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*, volume 8600 of *Lecture Notes in Computer Science*, pages 37–48. Springer, Berlin, 2014.
33. B. Q. Pinto, C. C. Ribeiro, I. Rosseti, and T. F. Noronha. Input data and detailed numerical results for 'A biased random-key genetic algorithm for routing and wavelength assignment under a sliding scheduled traffic model in WDM optical networks'. https://data.mendeley.com/datasets/r76d3pjnnk/, 2018. Last accessed on October 17, 2018.
34. R. Ramaswami and K. N. Sivarajan. Routing and wavelength assignment in all-optical networks. *IEEE/ACM Transactions on Networking*, 3:489–500, 1995.
35. M. G. C. Resende and C. C. Ribeiro. Restart strategies for GRASP with path-relinking heuristics. *Optimization Letters*, 5:467–478, 2011.
36. 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*, pages 483–514, Denver, 2016. Association for Computing Machinery.
37. M. G. C. Resende and C. C. Ribeiro. *Optimization by GRASP*. Springer, Boston, 2016.
38. A. Rohatgi. *WebPlotDigitizer (Version 4.1)*, 2018. Last visit in October 6, 2018.
39. C. V. Saradhi and M. Gurusamy. Scheduling and routing of sliding scheduled lightpath demands in WDM optical networks. In *Conference on Optical Fiber Communication and the National Fiber Optic Engineers Conference*, pages 1–3, Anaheim, 2007.

40. C. V. Saradhi, M. Gurusamy, and R. Piesiewicz. Routing fault-tolerant sliding scheduled traffic in WDM optical mesh networks. In *5th International Conference on Broadband Communications, Networks and Systems*, pages 197–202, 2008.
41. N. Skorin-Kapov. Heuristic algorithms for the routing and wavelength assignment of scheduled lightpath demands in optical networks. *IEEE Journal on Selected Areas in Communications*, 24:2–15, 2006.
42. N. Skorin-Kapov. Routing and wavelength assigment in optical networks using bin packing based algorithms. *European Journal of Operational Research*, 177:1167–1179, 2007.
43. W. Spears and K. A. De Jong. On the virtues of parameterized uniform crossover. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, San Mateo, 1991. Morgan Kaufman.
44. R. F. Toso and M. G. C. Resende. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30:81–93, 2015.
45. B. Wang, T. Li, X. Luo, and Y. Fan. Traffic grooming under a sliding scheduled traffic model in WDM optical networks. In *IEEE Workshop on Traffic Grooming in WDM Networks*, San Jose, 2004.
46. B. Wang, T. Li, X. Luo, Y. Fan, and C. Xin. On service provisioning under a scheduled traffic model in reconfigurable WDM optical networks. In *2nd International Conference on Broadband Networks*, pages 13–22, Boston, 2005.
47. H. Zang, J. P. Jue, and B. Mukherjeey. A review of routing and wavelength assignment approaches for wavelength-routed optical wdm networks. *Optical Networks Magazine*, 1:47–60, 2000.