### UNIVERSIDADE FEDERAL FLUMINENSE

## JULLIANY SALES BRANDÃO

# Algoritmos Genéticos com Chaves Aleatórias Tendenciosas para Problemas de Otimização em Redes

NITERÓI 2015

### UNIVERSIDADE FEDERAL FLUMINENSE

### JULLIANY SALES BRANDÃO

# Algoritmos Genéticos com Chaves Aleatórias Tendenciosas para Problemas de Otimização em Redes

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: Otimização Combinatória

Orientador: Celso Carneiro Ribeiro

Co-orientador: Thiago Ferreira de Noronha

NITERÓI

2015

#### Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

B817 Brandão, Julliany Sales Algoritmos genéticos com chaves aleatórias tendenciosas para problemas de otimização em redes / Julliany Sales Brandão. – Niterói, RJ : [s.n.], 2015. 111 f.
Tese (Doutorado em Computação) - Universidade Federal Fluminense, 2015. Orientadores: Celso Carneiro Ribeiro, Thiago Ferreira de Noronha.
1. Algoritmo genético. 2. Metaheurística. 3. Otimização combinatória (Computação). I. Título.

#### Julliany Sales Brandão

Algoritmos Genéticos com Chaves Aleatórias Tendenciosas para Problemas de Otimização em Redes

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: Otimização Combinatória

#### BANCA EXAMINADORA

Prof. Celso Carneiro Ribeiro - Orientador, UFF

Prof. Thiago Ferreira de Noronha - Co-orientador, UFMG

Profa. Maria Cristina Silva Boeres, UFF

Prof. Mauricio Guilherme de Carvalho Resende, Amazon

Profa. Noemi de La Rocque Rodriguez, PUC-Rio

Profa. Simone de Lima Martins, UFF

"A persistência é o caminho do êxito". (Charles Chaplin)

A Deus, aos meus pais Suelene e Valmir, ao meu marido Felipe, a minha irmã Susany, ao meu afilhado Lucas e aos meus avôs Antônio (in memoriam) e Domingos. Dedico.

# Agradecimentos

Agradeço a Deus, Nossa Senhora Aparecida e Santa Rita por me fortalecer na fé.

Aos meus pais e minha irmã Susany, 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!

Ao meu marido, por sua compreensão nos diversos momentos em que não pude lhe dá a devida atenção. Mesmo assim, sempre me incentivou e esteve ao meu lado me apoiando e me incentivando. Você é um presente de Deus para mim. Obrigada por tudo!

Ao meu sobrinho Lucas por me presentear com um sorriso e um "eu te amo, Juju"quando eu estava no auge do cansaço achando que nada mais daria certo.

Aos meus amigos e familiares que sempre torceram e oraram por mim, em especial, Dalva, Isaura, Ivonaldo, Zezé, Eduardo, Alessandra, Idário, Paula, Sandra e Jorge.

Aos meus amigos da UFF que ganhei durante o doutorado e que levarei para a vida inteira. Agradeço em especial, ao Eyder, Igor, Sabir e Pablo pelas discussões e apoio nos momentos de dificuldades.

Aos meus orientadores Celso Ribeiro e Thiago Noronha por extraírem o meu melhor. Tenho muito orgulho de ter trabalhado com cientistas admiráveis como vocês.

Ao professor Mauricio Resende por sua atenção, acolhida, amizade e ensinamentos durante o meu estágio na AT&T Labs Research.

À Lucia Resende por seu carinho maternal, incentivo e conselhos que muito me ajudaram durante a minha estadia nos Estados Unidos.

À Luzia Sergina por me receber atenciosamente em sua casa durante minhas idas para Belo Horizonte e por está sempre na torcida pelo meu sucesso.

À Teresa Cancela por sua atenção e amizade.

Ao Elbio Abib pelas discussões e atenção.

Aos amigos do CEFET-RJ que me apoiaram desde o início, em especial, Enoch, David, Cesar, William, Caleb, Rodrigo Callado, Raimundo Moraes, Marcelo Duarte, Carmen Perrota, Gisele Vieira, Leonardo Lima, Álvaro Chrispim e Luís Fonseca.

Ao diretor geral do CEFET-RJ Carlos Henrique Figueiredo e ao vice-diretor Mauricio Saldanha pelo apoio e autorização do meu afastamento das atividades na instituição, para que eu pudesse me dedicar exclusivamente ao doutorado, permitindo com isto um melhor rendimento e aprendizado durante o curso.

À CAPES pelo apoio financeiro que viabilizou meu estágio na categoria doutorado sanduíche na AT&T Labs Research nos Estados Unidos.

## Resumo

Esta tese relata resultados apresentados na forma de uma coletânea de artigos sobre aplicações de algoritmos genéticos com chaves aleatórias tendenciosas - BRKGA (do inglês *Biased Random Key Genetic Algorithm*) em diferentes problemas de otimização em redes. A motivação do uso desta técnica consiste nos bons resultados já apresentados na literatura para diversos problemas de otimização combinatória. Os três problemas estudados são: maximização de requisições atendidas em redes óticas, escalonamento de cargas divisíveis em um único período e escalonamento de cargas divisíveis em múltiplos períodos.

O problema de roteamento e atribuição de comprimentos de onda em redes WDM consiste em atribuir uma rota e um comprimento de onda a um conjunto de requisições de caminhos óticos, de modo que os caminhos óticos cujas rotas compartilham alguma fibra ótica usem comprimentos de onda diferentes e o número de requisições atendidas seja maximizado (max-RWA). Foram propostas seis heurísticas construtivas e um BRKGA. Três heurísticas foram baseadas em algoritmos para o problema de empacotamento e as outras três baseadas no problema de escalonamento em múltiplos processadores. Os experimentos computacionais mostraram que a heurística construtiva SPT, proposta nesta tese, obteve melhores resultados do que a melhor heurística construtiva da literatura. Além disso, o BRKGA desenvolvido nesta tese é a primeira heurística a tratar instâncias com mais de 27 nós. Ele encontrou soluções cujo desvio em relação ao valor ótimo foi, na média, 3,06%.

O problema de escalonamento de cargas divisíveis em um único período com processadores dedicados e heterogêneos consiste em selecionar um subconjunto de processadores, definir a ordem na qual os fragmentos de carga serão transmitidos para cada um deles e decidir a quantidade de carga a ser processada por cada processador. O objetivo é minimizar o makespan. Um BRKGA foi proposto nesta tese para o problema e obteve melhores resultados do que a melhor heurística da literatura. Experimentos computacionais mostraram que, para as 720 instâncias testes com até 160 processadores, o algoritmo proposto, BRKGA-DLS, encontrou soluções ótimas para 413 instâncias (dos 497 casos em que a solução ótima é conhecida). Já a melhor heurística da literatura, HeuRet, encontrou soluções ótimas para apenas 320 instâncias. Foi proposto um novo conjunto de instâncias, maiores e mais realistas, com 320 processadores. O BRKGA-DLS encontrou valores de soluções, em média, 2,38% melhores do que HeuRet.

No problema de escalonamento de cargas divisíveis em múltiplos períodos, a carga é distribuída para os processadores ativos em vários períodos, reduzindo o tempo ocioso em cada processador e permitindo uma melhor utilização do recursos. O objetivo é minimizar o *makespan*. Nesta tese foi proposto um BRKGA que obteve resultados 11,68% melhores do que o melhor algoritmo da literatura, em termos da qualidade média da solução.

**Palavras-chave**: Algoritmos genéticos com chaves aleatórias tendeciosas, otimização em redes, metaheurística, maximização de demandas, escalonamento de cargas divisíveis

# Lista de Figuras

1.1	Pseudocódigo básico de um algoritmo genético	2
1.2	Combinação uniforme parametrizada	3
1.3	Ilustração do processo de transição entre duas gerações	4
1.4	Pseudocódigo básico de um BRKGA	5

# Sumário

1	l Introdução			
	1.1	Algoritmos genéticos com chaves aleatórias tendeciosas	1	
	1.2	Aplicações de BRKGAs	6	
	1.3	Estrutura do trabalho	9	
2	Prol	olema de maximização de requisições atendidas em redes óticas	10	
	2.1	Trabalhos relacionados	11	
		2.1.1 Algoritmos e formulações exatas para max-RWA	11	
		2.1.2 Heurísticas para max-RWA	12	
	2.2	Contribuições	13	
3	Prol	olema de escalonamento de cargas divisíveis em um único período	15	
	3.1	Trabalhos relacionados	16	
	3.2	Contribuições	18	
4 Problema de escalonamento de cargas divisíveis em múltiplos períodos		olema de escalonamento de cargas divisíveis em múltiplos períodos	19	
	4.1	Trabalhos relacionados	20	
	4.2	Contribuições	22	
5	Con	clusões	23	
Re	eferên	cias	<b>24</b>	

Anexo A - 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 (2015), p. 1 - 23.

- Anexo B Brandão, J.S., Noronha, T.F., Resende, M.G.C., Ribeiro, C.C. "A biased random-key genetic algorithm for scheduling divisible loads". In Proceedings of the Multidisciplinary International Scheduling Conference: Theory and Applications (Prague, 2015), p. 570 - 578.
- Anexo C 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 Operational Research 22 (2015), p. 823 – 839.
- Anexo D 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". A ser submetido para Journal of Scheduling.

## Capítulo 1

# Introdução

## 1.1 Algoritmos genéticos com chaves aleatórias tendeciosas

Os algoritmos genéticos clássicos foram propostos em 1975 por John Holland [36], baseados na teoria da evolução de Darwin, onde os indivíduos mais aptos possuem uma probabilidade maior de se reproduzir. Na terminologia básica dos algoritmos genéticos, um indivíduo c é um vetor de n componentes. Cada componente  $c_i$ , para i = 1, ..., n é chamado gene e o seu valor é chamado alelo. Os indivíduos são associados a possíveis soluções de um problema. Para cada indivíduo é aplicada uma função de avaliação que corresponde a seu valor de aptidão, isto é, a capacidade do indivíduo em resolver esse problema.

Um algoritmo genético (AG) evolui um conjunto de indivíduos que compõem uma população P, ao longo de um certo número de gerações. A cada geração uma nova população é criada, utilizando os operadores genéticos de cruzamento e mutação. No cruzamento, os indivíduos da população corrente são combinados para produzir novos indivíduos para a próxima geração. O operador de mutação altera aleatoriamente um ou mais genes de um certo número de indivíduos. O algoritmo é repetido até que um critério de parada seja alcançado.

O pseudocódigo de um AG típico é apresentado na Figura 1.1. Na linha 1 a população P é inicializada. A sua evolução ocorre nas linhas 2 a 8. Na linha 3 são calculados os valores de aptidão de todos os indivíduos. Com base nestas aptidões, na linha 4 são selecionados os indivíduos pais que participarão do processo de geração dos descendentes através das operações de cruzamento e mutação, conforme indicado nas linhas 5 e 6, respectivamente. Na linha 7, a nova população é atualizada. Este procedimento é realizado até que um critério de parada seja alcançado. Por fim, na linha 9, o melhor indivíduo da população é retornado.

Procedimento GA			
1 Inicialize a população inicial $P$ ;			
2 enquanto Critério de parada não for satisfeito faça			
3 Calcule o valor de aptidão de cada indivíduo em $P$ ;			
4 Selecione os indivíduos para gerar uma nova população;			
5 Aplique o operador genético de cruzamento;			
6 Aplique o operador genético de mutação;			
7 Atualize a população;			
s fim			
9 Retorne o melhor indivíduo da população			

Figura 1.1: Pseudocódigo básico de um algoritmo genético

Algoritmos genéticos com chaves aleatórias - RKGA (do inglês *Random Key Genetic Algorithm*) foram introduzidos em 1994 por Bean [2] para resolver um problema de escalonamento de máquinas. Em um RKGA os indivíduos são representados por um vetor de números reais no intervalo contínuo [0, 1). Cada elemento do vetor é chamado de chave e é gerado aleatoriamente na população inicial. Daí vem o nome *random keys*.

A população é particionada em dois subconjuntos, um formado pelos indivíduos mais aptos da população, denominado conjunto elite, e o outro formado pelos indivíduos remanescentes, denominado conjunto não-elite.

Um algoritmo determinístico chamado *decodificador* mapeia um vetor de chaves aleatórias numa solução do problema de otimização. O custo desta solução é usado como valor de aptidão. O decodificador pode ser feito de várias maneiras. Os alelos podem ser utilizados como valores indicadores, para induzir permutações, entre outras possibilidades.

Em um RKGA, a mutação é introduzida utilizando o conceito de *mutantes* na população. *Mutantes* são vetores de chaves aleatórias gerados da mesma maneira que a população inicial. Em cada geração um pequeno número de *mutantes* são introduzidos na população. Eles previnem a convergência prematura do método para ótimos locais.

Os RKGAs utilizam a combinação uniforme parametrizada de Spears e DeJong [62] para a operação de cruzamento, combinando dois indivíduos escolhidos aleatoriamente na população. Seja n o número de genes dos indivíduos. Dados dois indivíduos  $c_1 e c_2$ , escolhidos aleatoriamente na população,  $p_a$  é a probabilidade de um indivíduo descendente  $c_{new}$  herdar um alelo do indivíduo  $c_1$ . O descendente  $c_{new}$  é gerado da seguinte forma. Para i = 1, ..., n, o seu *i*-ésimo alelo  $c_{new}(i)$  recebe a i-ésima chave do indivíduo  $c_1$  com uma probabilidade  $p_a$  e do indivíduo  $c_2$  com probabilidade  $1 - p_a$ .

A Figura 1.2 ilustra o processo de cruzamento de dois indivíduos com quatro genes cada um. Neste exemplo, o valor de  $p_a$  é igual a 0,7. Um número real é gerado aleatoriamente no intervalo [0,1). Se o valor gerado for menor ou igual a 0,7, então o descendente herda o alelo do indivíduo 1. Caso contrário, ele herda do indivíduo 2. Neste exemplo, o descendente herdou o alelo do indivíduo 1 do primeiro, terceiro e quarto genes, se assemelhando mais a ele do que ao indivíduo 2.

Indivíduo 1	0,32	0,77	0,53	0,85	$\vdash$
Indivíduo 2	0,26	0,15	0,91	0,44	
Número aleatório	0,58	0,89	0,68	0,25	`~ X
Relação com a probabilidade de cruzamento de 0,7	<	>	<	<	Cruzamento
Indivíduo descendente	0,32	0,15	0,53	0,85	-

Figura 1.2: Combinação uniforme parametrizada

Os algoritmos genéticos com chaves aleatórias tendenciosas - BRKGA surgiram em 2002 [27, 33, 34] e se diferem do RKGA na maneira como os indivíduos são selecionados para o cruzamento, vide Gonçalves e Resende [30] para uma revisão. Em um BRKGA um indivíduo é selecionado aleatoriamente do conjunto elite na população corrente e o outro do conjunto não-elite. Contrariamente, num RKGA, a seleção dos dois indivíduos é realizada aleatoriamente na população. Esta diferença entre os dois algoritmos faz com que BRKGAs tenham um melhor desempenho do que RKGAs. Em ambos algoritmos, os indivíduos podem ser selecionados para cruzamento mais de uma vez em uma mesma geração.

Um BRKGA evolui uma população de vetores de chaves aleatórias ao longo de um número de gerações. A Figura 1.3 ilustra a transição entre duas gerações de BRKGA. O lado esquerdo da figura representa a população corrente, particionada em dois subconjuntos:  $TOP \in REST$ , onde TOP refere-se aos indivíduos do conjunto elite e REST aos indivíduos remanescentes. O tamanho da população é |TOP| + |REST|. Os indivíduos são ordenados pelo seu valor de aptidão. O conjunto TOP contém os indivíduos mais aptos da população e o conjunto REST contém os indivíduos remanescentes. O conjunto REST contém os indivíduos remanescentes o conjunto REST contém os indivíduos remanescentes o conjunto REST contém os indivíduos contexes o conjunto REST contém os indivíduos contexes o conjunto REST contém os indivíduos contexes o conjunto REST contém os in

to REST é formado por dois subconjuntos disjuntos: MID e BOT, com o subconjunto BOT formado pelos piores elementos da população. A população da nova geração é criada da seguinte maneira: uma estratégia elitista é adotada e os indivíduos do conjunto TOP são copiados sem alterações para a próxima geração. Um número |BOT| de indivíduos mutantes são gerados aleatoriamente e |MID| = |REST| - |BOT| indivíduos são criados por cruzamento entre um indivíduo escolhido aleatoriamente de TOP e outro de REST. A população da nova geração é então ordenada de acordo com os valores de aptidões de seus indivíduos. Um indivíduo elite da geração anterior pode não ser elite na geração atual.



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

BRKGAs também utilizam a combinação uniforme parametrizada de Spears e De-Jong [62]. A Tabela 1.1 lista os intervalos de valores dos parâmetros de um BRKGA que devem ser ajustados, conforme recomendados por [30] e [53]. Entre os parâmetros específicos do problema, estão a representação da solução, o processo de decodificação e os critérios de parada.

Tabela 1.1:	Parâmetros	recomendados	

Parâmetro	Descrição	Valor recomendado
P	Tamanho da população	$ P  = a \cdot n$ , onde $a \ge 1$ é uma
		constante e $n$ é o tamanho do
		indivíduo
TOP	Tamanho da população elite	$0, 10 \cdot  P  \le  TOP  \le 0, 25 \cdot  P $
BOT	Tamanho da população de mutantes	$0,05 \cdot  P  \le  BOT  \le 0,30 \cdot  P $
$p_a$	Probabilidade de herdar um alelo elite	$0, 5 \le p_a \le 0, 8$

O pseudocódigo do BRKGA é apresentado na Figura 1.4. Na linha 1 a população é inicializada. A sua evolução ocorre nas linhas 2 a 10. Na linha 3 todos os indivíduos são

decodificados e seus valores de aptidão calculados. Na linha 4 a população é ordenada em ordem não-decrescente dos seus valores de aptidão. A população P é então particionada em dois subconjuntos: TOP e REST. O subconjunto TOP contém os indivíduos mais aptos da população, enquanto o subconjunto REST é formado pelos indivíduos remanescentes. Na linha 6, a próxima população é inicializada com os indivíduos do conjunto TOP da população atual. Na linha 7, são gerados aleatoriamente |BOT| indivíduos mutantes para a próxima população. Na linha 8, são criados |P| - |TOP| - |BOT| indivíduos por combinação uniforme parametrizada para a próxima população, onde um indivíduo é escolhido do conjunto TOP e outro do conjunto REST. Na linha 9, a próxima população é então ordenada de acordo com os valores de aptidão de seus indivíduos. Este procedimento é realizado até um critério de parada ser atingido. Este critério pode ser por exemplo, o número de populações evoluídas, o tempo total de execução, a qualidade da melhor solução encontrada, entre outros. Por fim, na linha 11, o melhor indivíduo

Procedimento	BRKGA
--------------	-------

1 Inicialize a população inicial P;

2 enquanto Critério de parada não for satisfeito faça

**3** Calcule o valor de aptidão de cada indivíduo em *P*;

4 Ordene a população P em ordem não-decrescente de seus valores de aptidão;

5 Particione P em dois conjuntos:  $TOP \in REST$ ;

- 6 Copie os indivíduos do conjunto *TOP* da população atual para a próxima população;
- 7 Gere aleatoriamente |BOT| indivíduos mutantes para a próxima população;
- 8 Crie |P| |TOP| |BOT| indivíduos por combinação uniforme parametrizada para a próxima população, selecionando um indivíduo de TOP e outro de REST;
- Ordene a próxima população de acordo com os valores de aptidão de seus indivíduos;

10 fim

11 **Retorne** o melhor indivíduo da população

Figura 1.4: Pseudocódigo básico de um BRKGA

Além dos métodos disponíveis na literatura, a performance do BRKGA será comparada com um procedimento multipartida. Este procedimento consiste em utilizar a cada iteração a mesma heurística de decodificação do BRKGA, com valores para as chaves gerados aleatoriamente. Ao contrário do BRKGA, nada é aprendido de uma geração para outra, pois o procedimento multipartida não realiza cruzamentos. Assim, ele pode ser implementado a partir da implementação do BRKGA, com o conjunto elite possuindo apenas um indivíduo, o melhor indivíduo conhecido até o momento, e os demais indivíduos da população sendo todos mutantes. O procedimento multipartida é aqui utilizado para avaliar a convergência do BRKGA.

### 1.2 Aplicações de BRKGAs

Em diversas aplicações, os BRKGAs têm apresentado soluções tão boas ou melhores que os RKGAs [32], e os algoritmos genéticos clássicos [30]. A seguir são apresentados alguns exemplos de sucesso dos BRKGAs.

O problema de atribuição de tráfego visa minimizar o atraso total de viagem entre todos os viajantes. O problema de preços de pedágio determina o conjunto de pedágios e tarifas correspondentes que beneficiariam coletivamente todos os viajantes. Buriol et al. [17] propuseram uma abordagem para resolver os dois problemas em conjunto, fazendo uso de um BRKGA para a otimização do desempenho da rede de transporte através da atribuição estratégica dos preços de pedágios em alguns dos pontos da rede rodoviária. Uma rede de transporte pode ter milhares de cruzamentos e centenas de segmentos de estrada, e o BKRGA leva vantagem nos mecanismos para acelerar os algoritmos de menores caminhos.

Linhas de montagem ou de fabricação são utilizadas para fabricar grandes quantidades de produtos padronizados. Uma linha de montagem é constituída de uma sequência de bestações de trabalho ligadas por uma correia de transporte, por meio da qual as unidades dos produtos passam. Cada estação de trabalho realiza um subconjunto das operações necessárias para a fabricação de q produtos. Cada unidade de produto permanece em uma estação por um tempo fixo, chamado tempo do ciclo. Em linhas de montagem tradicionais, estações de trabalho são consecutivamente dispostas em uma linha reta. Cada unidade do produto continua ao longo desta linha e passa uma vez por cada estação de trabalho. A principal decisão consiste em definir uma atribuição de operações para as estações de trabalho de modo que a eficiência da linha seja maximizada. Gonçalves e Almeida [33] descreveram um BRKGA para balanceamento de linha de montagem. Para mostrar a robustez do método, os autores apresentaram os resultados computacionais utilizando três conjuntos de instâncias testes encontradas na literatura, que juntos totalizam 269 instâncias. Dois experimentos foram realizados. No primeiro, comparou-se um BRKGA com a heurística EUREKA de [35] e depois com as heurísticas de busca tabu PrioTabu e EurTabu de [57]. O BRKGA encontrou soluções tão boas quanto as encontradas por EUREKA e foi melhor em uma. Ele também obteve um número 7% maior de melhores

soluções que PrioTabu e o mesmo número de melhores soluções que EurTabu.

Em um problema de escalonamento de projeto, várias atividades devem ser programadas. Relações de precedência entre as atividades restringem o início de uma atividade para após a conclusão da outra. O objetivo é minimizar o makespan, isto é, o tempo de conclusão da última atividade programada. Quando as atividades exigem recursos com capacidades limitadas têm-se um problema de escalonamento de projeto com recursos limitados - RCPS. Gonçalves et al. [31] descreveram um BRKGA para o RCPSP e consideraram um total de 600 instâncias do conjunto de problemas testes padrão J120. Neste conjunto teste cada instância tem 120 atividades e requer quatro tipos de recursos. O BRKGA foi comparado com diversos métodos da literatura para o problema e obteve melhores resultados sobre todos. A aproximação de Debels et al. [20] foi a que obteve uma performance mais similar. O BRKGA conseguiu melhorar as melhores soluções conhecidas para 11 instâncias do repositório de problemas testes.

No problema *job-shop scheduling* - JSP, um conjunto finito de tarefas é composto por várias operações. Estas operações devem ser processadas em um conjunto finito de máquinas. Cada operação usa uma máquina por um período de tempo fixo e cada máquina pode processar uma operação por vez. Depois que o processamento na máquina se inicia, ele não pode ser interrompido até seu término. As operações de uma dada máquina têm que ser processadas em uma dada ordem. O JSP consiste em encontrar um escalonamento de operações nas máquinas que minimize o *makespan*. Gonçalves et al. [29] apresentaram um BRKGA para o problema e testaram em um conjunto de instâncias clássicas da literatura. Das 43 instâncias testadas, todas foram resolvidas por BRKGA e em 31 delas (72% dos problemas) ele encontrou a melhor solução conhecida e obteve um desvio relativo médio das melhores soluções conhecidas de 0,39%.

O problema de roteamento e atribuição de comprimento de ondas - RWA em redes óticas WDM consiste em rotear um conjunto de caminhos óticos e atribuir um comprimento de onda para cada um deles tal que para os caminhos óticos que compartilham alguma aresta são atribuídos comprimentos de onda diferentes. Noronha et al. [53] propuseram um BRKGA para a versão do problema 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 uma variante multipartida da heurística BFD-RWA [61], MS-RWA, assim como com a heurística busca tabu 2-EDR+TS-PCP [54]. Os autores observaram que o BRKGA relações entre as chaves e boas soluções, convergindo para melhores soluções, em média, em 23% menos tempo que MS-RWA. O desvio médio das soluções obtidas por BRKGA foi quase 50% menor que o apresentado pela busca tabu.

Outras aplicações de BRKGAs são apresentadas em [30]. As aplicações descritas a seguir são as principais contribuições dessa tese.

Um BRKGA para uma outra variante do problema RWA, denominada max-RWA, foi proposto no decorrer desta tese e publicado em [15, 16]. Neste caso, dado um número fixo e pré-determinado de comprimentos de ondas, o objetivo é maximizar o número de requisições atendidas. O BRKGA foi comparado com o método exato de Martins et al. [51, 52] e com a busca tabu de Dzongang et al. [26]. Ele encontrou soluções próximas aos limites superiores de [51, 52] e foi, na média, melhor que [26]. O Capítulo 2 define o problema e os resultados alcançados são apresentados no artigo que compõe o Anexo A.

Uma carga divisível é uma carga que pode ser dividida arbitrariamente sem restrições de precedência. O problema de escalonamento de cargas divisíveis em único período consiste em selecionar um subconjunto de processadores, definir a ordem na qual os fragmentos de carga serão transmitidos para cada um deles e decidir a quantidade de carga que será processada por cada processador com o objetivo de minimizar o *makespan*. Como parte desta tese, foi proposto um BRKGA para resolver este problema, publicado em [13, 14]. Os resultados mostraram que o algoritmo genético foi melhor que HeuRet [1], a melhor heurística conhecida na literatura. O problema é definido no Capítulo 3 e os resultados alcançados são apresentados nos artigos que compõem os Anexos B e C.

Também nesta tese, foi tratado o problema de escalonamento de cargas divisíveis em múltiplos períodos. Neste caso, os processadores ativos recebem fragmentos de carga em mais de uma vez, permitindo um melhor aproveitamento dos recursos e a redução do *makespan*. Esta é uma versão mais difícil do problema pois, além das variáveis já envolvidas e tratadas em [13, 14], é necessário determinar também a quantidade de períodos e suas respectivas durações. Para solucionar o problema, foi proposto um BRKGA e os resultados obtidos foram comparados com aqueles encontrados pelo melhor algoritmo da literatura [60] e com uma heurística multipartida. O BRKGA, denominado GA-KEY, foi melhor do que os demais métodos, em todas as instâncias testadas. O Capítulo 4 define o problema e os resultados alcançados são apresentados no artigo que compõe o Anexo D.

### 1.3 Estrutura do trabalho

Esta tese apresenta a aplicação dos algoritmos genéticos com chaves aleatórias tendenciosas para três problemas de otimização em redes e está organizada como segue: o Capítulo 2 apresenta o problema de maximização de requisições atendidas em redes óticas, max-RWA. O Capítulo 3 define o problema de escalonamento de cargas divisíveis em um único período. Já a versão deste problema em múltiplos períodos é definida no Capítulo 4. Por fim, as conclusões são apresentadas no Capítulo 5.

# Capítulo 2

# Problema de maximização de requisições atendidas em redes óticas

As informações são transmitidas em redes óticas através de fibras óticas. Cada enlace de fibra ótica opera a taxas de terabits por segundo, o que é muito mais rápido do que os dispositivos eletrônicos utilizados para transmissão e recepção de dados. A tecnologia de multiplexação de comprimentos de onda (do inglês Wavelength Division Multiplexing - WDM) permite que dezenas de conexões sejam transmitidas simultaneamente em um mesmo sinal luminoso, cada uma delas codificada em uma faixa de frequência diferente da luz. Quando o sinal chega na extremidade da fibra ótica, os comprimentos de onda são separados. Em seguida, caso a conexão correspondente esteja deixando a rede, o comprimento de onda é convertido em um sinal elétrico e caso a conexão correspondente esteja seguindo em direção a outro nó da rede, o comprimento de onda é multiplexado novamente em outra fibra ótica. Caminhos óticos são conexões entre origens e destinos onde não há conversão do sinal ótico para o domínio eletrônico nos nós intermediários, sem que ocorram os atrasos causados por estas transformações. Cada caminho ótico é caracterizado por uma rota e um comprimento de onda com o qual é multiplexado. Dois caminhos óticos podem utilizar o mesmo comprimento de onda desde que não compartilhem nenhum enlace.

Dados uma topologia física de uma rede ótica e um conjunto de requisições de caminhos óticos que devem ser estabelecidos definindo a topologia lógica desta rede, o problema de roteamento e atribuição de comprimentos de onda em redes WDM (*Routing and Wavelength Assignment* - RWA, em inglês) consiste em rotear o conjunto de caminhos óticos e atribuir um comprimento de onda para cada um deles, de modo que caminhos óticos que compartilhem algum enlace da rede usem comprimentos de onda diferentes. Versões diferentes de RWA são caracterizadas por diferentes critérios de otimização e padrões de tráfego [19, 70]. São consideradas variantes nas quais as requisições de caminho ótico são conhecidas de antemão, não há conversão de comprimento de onda, isto é, o caminho ótico deve ser atribuído ao mesmo comprimento de onda em todos os enlaces de fibras óticas ao longo de seu percurso e o tráfego é assimétrico (se existe um caminho ótico de um nó u para um nó v, não necessariamente existe um caminho ótico de v para u, nem que este último tem que usar a mesma rota ou o mesmo comprimento de onda do primeiro).

Seja G = (V, A) um grafo direcionado representando a topologia física da rede, onde V é o conjunto de nós e A é o conjunto de arcos (cada arco corresponde a um enlace de fibras óticas). Seja também R o conjunto de requisições de caminhos óticos, cada um definido por um nó origem e um destino em V. Denota-se por  $\lambda$  o número de comprimentos de onda disponíveis. Na variante min-RWA do problema [28], o número de comprimentos de onda disponíveis é ilimitado e o objetivo é minimizar a quantidade utilizada para estabelecer todas as requisições de caminhos óticos em R. Este trabalho se concentra na variante max-RWA [18], onde  $\lambda < |R|$ . Portanto, pode não ser possível aceitar todas as requisições em R. O objetivo é maximizar o número de requisições que podem ser atendidas. As variantes min-RWA e max-RWA do problema foram provadas ser NP-difícil em [28] e [18], respectivamente.

### 2.1 Trabalhos relacionados

#### 2.1.1 Algoritmos e formulações exatas para max-RWA

A maior parte da literatura sobre algoritmos para max-RWA trata de formulações de programação inteira [40, 41, 44, 45, 52, 55]. Revisões das formulações para max-RWA podem ser encontradas em [38, 39, 51].

Krishnaswamy e Sivarajan [44] desenvolveram uma formulação compacta baseada em arcos para max-RWA. Esta formulação permite ciclos, mas eles argumentaram que esses ciclos não têm qualquer impacto no valor da função objetivo. A formulação compacta com ciclos foi baseada em uma anterior proposta em [41]. Esta formulação não melhora os limites superiores de [41], mas pode ser resolvida de maneira mais eficiente por resolvedores de programação inteira mista. Martins [51] melhorou as formulações compactas de [41, 44] e encontrou soluções ótimas para instâncias com até 18 nós.

Ramaswami e Sivarajan [55] propuseram uma formulação baseada em conjuntos independentes com um número exponencial de variáveis. Eles provaram que os limites superiores encontrados pela sua relaxação linear nunca são menores que os das melhores formulações compactas de programação inteira encontradas na literatura, até o momento da sua publicação. Jaumard et al. [40] implementaram um algoritmo de geração de colunas baseado nesta formulação, que resolveu até a otimalidade instâncias com até 27 nós.

Lee et al. [45] apresentaram uma formulação com base no problema de configuração de roteamento independente com restrições de cardinalidades [48], no qual a relaxação linear pode ser resolvida por um algoritmo de geração de colunas. Jaumard et al. [40, 41] propuseram uma melhoria nesta formulação usando configuração de roteamento independente maximal e encontraram limites superiores melhores do que aqueles de [45] para instâncias com até 27 nós.

Martins et al. [51, 52] propuseram um nova e melhorada formulação baseada em [40, 45, 55]. Esta abordagem obteve os primeiros limites superiores para três instâncias baseadas na rede ATT2 com até 71 nós, que não foram possíveis ser calculados nos trabalhos anteriores da literatura por falta de memória disponível. Eles encontraram os melhores limites superiores para os maiores exemplos na literatura de max-RWA com até 90 nós.

### 2.1.2 Heurísticas para max-RWA

A complexidade no pior caso dos algoritmos baseados em formulações de programação inteira propostos na literatura cresce exponencialmente com o tamanho da rede. Com isso, apenas para instâncias pequenas as soluções ótimas são conhecidas. O estado da arte destes algoritmos só fornece limites superiores para os valores ótimos das maiores instâncias da literatura. Até o momento deste trabalho, sabe-se que as poucas heurísticas existentes para max-RWA são algoritmos gulosos [44, 50], uma abordagem de particionamento de grafos em [7] e uma heurística de busca tabu em [26].

Krishnaswamy e Sivarajan [44] propuseram duas heurísticas de arredondamento baseadas na relaxação linear de suas formulações de programação inteira. Experimentos computacionais realizados em duas redes com 14 e 20 nós mostraram que o gaps de otimalidade relativos médios para a melhor das duas heurísticas foram de 6,0% e 7,2%, respectivamente.

Manohar et al. [50] desenvolveram a heurística Greedy-EDP-RWA. A cada iteração um subconjunto de caminhos óticos é selecionado e roteado com arcos disjuntos pela heurística BGAforEDP para o problema de encontrar o maior subconjunto de caminhos óticos com arcos disjuntos - max-EDP [43]. Em seguida, todos os caminhos óticos neste subconjunto são atribuídos ao mesmo comprimento de onda e o procedimento é repetido com os caminhos óticos restantes. Esta heurística foi proposta para a variante do problema min-RWA, mas os autores argumentaram que ela também pode ser usada para max-RWA, executando BGAforEDP por um certo número de iterações. Os autores mostraram que o algoritmo proposto por eles foi mais rápido e encontrou soluções tão boas quanto os algoritmos de programação linear para min-RWA, até o momento de sua publicação. Não há experimentos computacionais relatados sobre o desempenho desta heurística para max-RWA.

Uma heurística de busca tabu foi proposta por Dzongang et al. [26]. Esta heurística possui dois passos principais. No primeiro passo, ela seleciona um conjunto de caminhos no grafo. Em seguida, estes caminhos selecionados são utilizados para construir a solução do problema, de modo que caminhos óticos que compartilhem o mesmo enlace não utilizem o mesmo comprimento de onda. Experimentos computacionais com as mesmas instâncias utilizadas em [44] mostraram que a busca tabu encontrou soluções muito melhores que as da heurística de arredondamento de [44], com gaps médios de otimalidade de 1,41% e 1,53% para as instâncias com 14 e 20 nós, respectivamente.

Belgacem e Puech [7] propuseram uma heurística de decomposição para max-RWA, na qual a instância original é particionada em instâncias menores e são resolvidas exatamente por programação inteira. As soluções locais são combinadas com uma solução viável. Esta proposta foi validada por uma aplicação na rede de transporte européia EBN57 com 57 nós e para redes planares com até 500 nós geradas aleatoriamente.

### 2.2 Contribuições

Os resultados computacionais são apresentados no artigo que compõe o Anexo A, onde é mostrado que a maioria dos trabalhos da literatura de max-RWA se baseia em formulações de programação inteira e se concentra em encontrar limites superiores para os valores ótimos. Somente pequenas ocorrências com não mais de 27 nós são resolvidos na otimalidade e apenas limites superiores são conhecidos para instâncias maiores, com até 90 nós.

Para resolver o problema de max-RWA foram propostas seis heurísticas construtivas e um algoritmo genético com chaves aleatórias tendenciosas - BRKGA. Três heurísticas foram baseadas em algoritmos para o problema de empacotamento e as outras três foram baseadas no problema de escalonamento em múltiplos processadores. Destas seis heurísticas construtivas propostas, a que encontrou as melhores soluções foi SPT. Ela é baseada no problema de escalonamento em múltiplos processadores. SPT inicialmente ordena as requisições de caminhos óticos em valores não-decrescentes de seus caminhos mínimos e resolve os empates arbitrariamente. Cada comprimento de onda é tratado como uma cópia do grafo G = (V, A) (um processador), disponível desde o início, e os caminhos óticos são tratados como processos que devem ser escalonados nestes processadores. Cada requisição de caminho ótico é associada a um valor definido como o número mínimo de arcos em um caminho entre origem e destino em G. Este valor é análogo ao tempo de execução de um processo no escalonamento em múltiplos processadores. A heurística retorna o número máximo de requisições atendidas.

A heurística SPT foi utilizada como decodificador pelo BRKGA e também embutida numa variante multipartida para comparação entre os métodos. O BRKGA também foi comparado com o método exato de Martins et al. [51, 52] e a busca tabu de Dzongang et al. [26]. Quatro classes de instâncias foram testadas. Nas três primeiras, comparou-se o BRKGA com o método exato e notou-se que o desvio médio não foi superior a 4% do limite superior. Quando comparado com a heurística multipartida em termos da qualidade da solução, em todos os casos testados, o BRKGA encontrou soluções melhores e conseguiu identificar boas relações entre soluções e chaves. Para o conjunto de instâncias tratadas pela busca tabu, o BRKGA obteve, em média, melhores resultados.

## Capítulo 3

# Problema de escalonamento de cargas divisíveis em um único período

Uma quantidade de carga divisível W é uma tarefa computacional que pode ser arbitrariamente dividida e distribuída entre diferentes processadores para ser processada em paralelo. Supõe-se que os processadores estejam dispostos numa topologia estrela e a carga é armazenada no processador central, chamado *mestre*. Este último divide a carga em fragmentos de tamanhos arbitrários e os transmite para os outros processadores, chamados *trabalhadores*. Assume-se que o mestre não processa nenhuma carga. No restante do texto o termo trabalhador é utilizado para referenciar cada processador, enquanto o termo mestre é usado para diferenciar o processador mestre dos demais.

O mestre só pode enviar carga para um trabalhador por vez. Qualquer trabalhador só pode iniciar o processamento depois de ter recebido completamente seu respectivo fragmento de carga (modo bloqueante). Os trabalhadores são heterogêneos em termos de poder de processamento, velocidade de comunicação e tempo de inicialização da comunicação com o mestre. Nem todos os processadores devem necessariamente ser utilizados para o processamento da carga. Consequentemente, como a carga está dividida, a escolha de quais os processadores serão utilizados e a ordem em que os fragmentos de carga serão transmitidos influenciam no tempo de processamento total da carga.

Este capítulo trata a mesma variante do problema de escalonamento de cargas divisíveis - DLSP (do inglês *Divisible Load Scheduling Problem*) estudada por [1, 12]. Seja  $W \ge 0$  a quantidade de carga a ser processada, 0 (zero) o índice do processador mestre, e  $P = \{1, ..., |P|\}$  o conjunto de índices dos processadores trabalhadores. Cada trabalhador  $i \in P$  tem (i) uma latência de comunicação  $g_i$  para estabelecer a comunicação com o mestre, (ii) um tempo de comunicação  $G_i$  necessário para receber cada uma das unidades de carga a partir do mestre e (iii) um tempo de processamento  $w_i$  utilizado para processar cada unidade de carga. Portanto, são necessárias  $g_i + \alpha_i \cdot G_i$  unidades de tempo para o mestre transmitir um fragmento de carga de tamanho  $\alpha_i$  para o trabalhador  $i \in P$ . Além disso, o trabalhador leva  $w_i \cdot \alpha_i$  unidades de tempo adicionais para processar o fragmento de carga que lhe é atribuído.

O problema de escalonamento de cargas divisíveis em um único período consiste em (a) definir a ordem em que os fragmentos de carga serão transmitidos para cada trabalhador, (b) selecionar um subconjunto  $A \subseteq P$  de trabalhadores, denominados trabalhadores ativos, e (c) definir a quantidade de carga  $\alpha_i$  que cada trabalhador  $i \in A$  irá processar, com  $\sum_{i=1}^{|P|} \alpha_i = W$ , de modo a minimizar o makespan, isto é, o tempo total decorrido desde o mestre começar a enviar carga para o primeiro trabalhador, até o último trabalhador terminar a computação. Este problema foi provado ser NP-difícil em [68]. Nota-se que  $\alpha_i = 0$ , para todo  $i \in P \setminus A$ .

Uma vez que a carga pode ser arbitrariamente dividida, todos os trabalhadores param ao mesmo tempo na solução ótima [6]. Se a ordem em que os fragmentos são transmitidos para os trabalhadores é fixa, então a melhor solução pode ser calculada em tempo O(|P|), utilizando o algoritmo AlgRap desenvolvido em [1]. Em outras palavras, dada uma permutação dos trabalhadores em P, AlgRap calcula o conjunto de trabalhadores ativos e a quantidade de carga que deve ser enviada para cada um deles de modo a minimizar o makespan. Portanto, o DLSP em único período pode ser reduzido ao problema de encontrar a melhor permutação dos processadores, ou seja, aquela que induz à solução com o makespan mínimo.

### 3.1 Trabalhos relacionados

Existem muitas variantes do DLSP na literatura. O escalonamento de carga divisível pode ser realizado em um *único período* ou em *múltiplos períodos*. Em um único período [1, 3, 4, 6, 9, 11, 12, 21, 24, 25, 42, 46, 59, 63, 64, 65] cada processador ativo recebe e processa um único fragmento de carga. No caso de múltiplos períodos [1, 4, 5, 8, 22, 23, 24, 25, 37, 47, 49, 58, 60, 65, 66, 67, 68, 69], cada processador ativo recebe e processa múltiplos fragmentos de carga.

Os processadores podem ser homogêneos ou heterogêneos. Se os processadores são homogêneos, os valores de  $g_i$ ,  $G_i \in w_i$  são os mesmos para todos os processadores  $i \in P$  [4, 10, 11, 42, 47, 49, 66, 67, 69]. Já no caso de processadores heterogêneos, os valores

de  $g_i$ ,  $G_i$  e  $w_i$  podem ser diferentes para cada processador [1, 3, 5, 6, 8, 12, 21, 22, 23, 24, 25, 37, 46, 58, 59, 60, 63, 64, 65, 66, 67, 68, 69]. É mais comum encontrar redes com sistemas heterogêneos.

O sistema pode ser dedicado ou não-dedicado. Quando o sistema é dedicado, assumese que todos os recursos (processadores, memória, rede e outros recursos) são utilizados para processar uma única carga computacional [1, 6, 8, 9, 11, 12, 21, 24, 25, 58, 63, 64, 65, 66, 67]. Um sistema não-dedicado pode ser usado para processar simultaneamente diferentes cargas computacionais [8, 9, 59]. A maioria dos autores assumem sistemas *dedicados*. Neste caso, uma estimativa dos recursos disponíveis é usada no processo de otimização.

Pode haver limitação de memória nos processadores, isto é, no tamanho máximo de fragmento de carga que pode ser recebido por cada processador. Quando tal limitação existe, o problema é dito com *memória restrita* [8, 9, 22, 24, 25, 46, 49, 65, 66, 67]. Caso contrário, o problema é dito *irrestrito* [1, 4, 6, 10, 11, 12, 21, 47, 58, 59, 60, 63, 64, 68]. Muitos autores assumem sistemas irrestritos, porque há geralmente uma abundância de memória secundária (por exemplo, unidades de disco rígido) para armazenar temporariamente o fragmento de carga do trabalhador.

Neste capítulo da tese é considerado o DLSP em único período de escalonamento, irrestrito, com processadores trabalhadores heterogêneos e dedicados [1, 4, 6, 12, 24, 64, 68], que foi provado ser NP-difícil em [68]. Blazewicz e Drozdowski [12] mostraram que uma vez que uma permutação dos trabalhadores é dada, uma solução com makespan mínimo pode ser obtida em tempo  $O(|P| \log |P|)$ . Mais tarde, Abib e Ribeiro [1] propuseram um algoritmo mais rápido, denominado AlgRap, que encontra esta solução em tempo O(|P|). Beaumont et al. [6] mostraram que se  $g_i = 0$ , para todo  $i \in P$ , então o DLSP em único período pode ser resolvido polinomialmente por ordenar os trabalhadores em ordem não-decrescente dos valores de  $G_i$ . Eles também mostraram que se  $G_i = G_j$ , para todo  $i, j \in P$ , então o problema pode ser resolvido ordenando os trabalhadores em ordem não-decrescente do produto de  $g_i \cdot w_i$ .

Uma formulação de programação não-linear para o DLSP em único período, sem restrições de memória, com processadores dedicados e heterogêneos foi proposta em [21]. Nesta formulação todos os processadores são utilizados. Em [65] foi proposta uma versão melhorada de [21], que permite que nem todos os processadores sejam utilizados. Além disso, nesta formulação é considerada a restrição de memória. No entanto, não há algoritmo eficiente na literatura para resolvê-las. A primeira formulação de programação linear inteira mista foi proposta em [1]. Experimentos computacionais relatados em [1] mostraram que, com base nesta formulação, o software CPLEX foi capaz de encontrar soluções ótimas para 490 dos 720 casos testes, com até 160 processadores. No entanto, para as instâncias consideradas difíceis, o CPLEX foi muito lento e não conseguiu provar a otimalidade. Isto motivou o desenvolvimento de heurísticas para resolver esta versão do DLSP.

A melhor heurística da literatura, até o momento, para a mesma variante DLSP estudada neste capítulo foi proposta por Abib e Ribeiro [1] e denominada HeuRet. A cada iteração, HeuRet estima o índice de desempenho  $e_i$  de cada trabalhador  $i \in P$  e constrói uma solução, com os processadores P em uma ordem não-decrescente dos valores de  $e_i$ . A heurística define  $e_i = G_i$ , para todo  $i \in P$ , na primeira iteração e faz uso do algoritmo exato AlgRap para calcular uma solução inicial  $s_0$  com makespan  $T_0$ . Em seguida, a cada nova iteração k, o desempenho estimado  $e_i$  de cada trabalhador  $i \in P$  é atualizado a partir dos valores de  $g_i$ ,  $w_i$ ,  $G_i \in T_{k-1}$ , e uma nova solução com makespan  $T_k$  é construída por AlgRap, considerando a nova ordem definida nos trabalhadores pelos índices de desempenho recém-atualizados. O procedimento para quando  $T_{k-1} < T_k$ , ou seja, quando a nova solução degenera o makespan da anterior, devido à utilização desnecessária de alguns processadores.

### 3.2 Contribuições

Para resolver este problema foi proposto um algoritmo genético com chaves aleatórias tendenciosas, denominado BRKGA-DLS, utilizando o algoritmo AlgRap como decodificador. Os resultados computacionais alcançados se encontram nos artigos que compõem os Anexos B e C.

O BRKGA proposto nesta tese obteve melhores resultados do que a melhor heurística da literatura, HeuRet. Experimentos computacionais mostraram que para as 720 instâncias testes com até 160 processadores, BRKGA-DLS encontrou soluções ótimas para 413 instâncias (dos 497 casos em que a solução ótima é conhecida). Já a heurística HeuRet encontrou soluções ótimas para apenas 320 instâncias. Foi proposto um novo conjunto de instâncias, maiores e mais realistas, com 320 processadores. O BRKGA-DLS encontrou valores de soluções, em média, 2,38% melhores do que HeuRet.

## Capítulo 4

# Problema de escalonamento de cargas divisíveis em múltiplos períodos

Este capítulo trata do problema de escalonamento de cargas divisíveis em múltiplos períodos, DLS-MR. Neste caso, os fragmentos de carga podem ser enviados para os processadores mais de uma vez. Isto reduz o tempo ocioso de cada processador e proporciona uma melhor utilização dos recursos para minimizar o *makespan*. Os processadores estão dispostos numa topologia estrela e a carga é armazenada em um processador central ou *mestre*. O mestre divide a carga em fragmentos de tamanhos arbitrários e transmite cada um deles para os processadores *trabalhadores*. Assume-se que o mestre só pode enviar carga a apenas um trabalhador por vez e que não participa do processamento. Além disso, um trabalhador só pode começar a processar seu fragmento de carga após tê-lo recebido integralmente (modo bloqueante).

Os trabalhadores são heterogêneos em termos de poder de processamento, velocidade de comunicação, tempo para iniciar a comunicação com o mestre e tempo para iniciar o processamento. Em razão dos tempos de inicialização, comunicação e processamento, poderá ser mais rápido não utilizar todos os processadores trabalhadores disponíveis para processar a carga. Os trabalhadores utilizados no processamento são denominados *trabalhadores ativos*. A ordem que o mestre usa para transmitir a carga para cada trabalhador é chamada de *ordem de transmissão*. Uma vez que esta ordem é definida, a transmissão é realizada em múltiplos períodos.

Nos sistemas em múltiplos períodos [4, 8, 10, 22, 23, 37, 47, 49, 58, 60, 65, 66, 67, 68, 69], cada trabalhador ativo recebe um fragmento de carga por período. A ordem de transmissão é a mesma para todos os períodos [66, 67, 69]. Estes sistemas podem reduzir o makespan, por diminuir os tempos ociosos, em adição aos repetidos custos de

inicialização.

O foco deste capítulo é o problema de escalonamento de cargas divisíveis em múltiplos períodos introduzido por [6, 60]. Seja  $W \ge 0$  a quantidade de carga a ser processada, 0 (zero) o índice do processador mestre, e  $P = \{1, ..., |P|\}$  o conjunto de índices dos processadores trabalhadores. Cada processador  $i \in P$  tem (i) um tempo de inicialização  $g_i$  para iniciar a comunicação com o mestre, (ii) um tempo de comunicação  $G_i$  necessário para receber cada unidade de carga do mestre, (iii) um tempo de inicialização  $s_i$  para iniciar a computação da carga e (iv) um tempo de processamento  $w_i$  necessário para processar cada unidade de carga. Portanto, são necessárias  $g_i + Gi \cdot \theta_i^j$  unidades de tempo para o mestre transmitir um fragmento de carga de tamanho  $\theta_i^j$  para o processador  $i \in P$ no período j. Além disso, este trabalhador leva  $s_i + w_i \cdot \theta_i^j$  unidades de tempo para processar o fragmento de carga atribuído a ele.

O problema de escalonamento de cargas divisíveis em múltiplos períodos (MR-DLS) consiste em (a) definir a ordem de transmissão  $\pi$ , (b) decidir o número n de trabalhadores ativos, (c) escolher o número m de períodos, e (d) decidir a quantidade de carga  $\theta_i^j$ que será transmitida para cada processador  $i \in P$  no período  $j \in \{1, \dots, m\}$ , com  $\sum_{i \in P} \sum_{j \in \{1, \dots, m\}} \theta_i^j = W$ , e assim minimizar o makespan, isto é, o tempo total gasto desde o mestre iniciar o envio de carga para o primeiro trabalhador, até o último trabalhador terminar a computação. Sem perda de generalidade e para facilitar a notação, assumese, no restante deste capítulo, que o processador  $i \in P$  é o *i*-ésimo trabalhador em  $\pi$ . Além disso, assume-se também que os primeiros n trabalhadores em  $\pi$  são ativos, isto é,  $A = \{i | i = 1 \dots n\}$ , e que  $\theta_i^j = 0$ , para todo  $i \in \{n + 1, \dots |P|\}$  e  $j \in \{1, \dots, m\}$ .

#### 4.1 Trabalhos relacionados

A variante do MR-DLS estudada neste trabalho é irrestrita, assumindo processadores trabalhadores dedicados e heterogêneos. Ela foi provada ser NP-difícil em [49, 68].

Yang et al. [67, 69] foram os primeiros a tratar problemas de escalonamento em múltiplos períodos com trabalhadores heterogêneos. Eles propuseram uma abordagem heurística chamada UMR. Os trabalhadores são ordenados em ordem não-decrescente dos valores de  $G_i$ . Esta ordem é utilizada como a ordem de transmissão. Seja a duração do período de um trabalhador *i* no período *j* a soma dos tempos de inicialização, de comunicação e computação de *i* no período *j*. UMR acrescenta uma restrição adicional de que o fragmento de carga que cada trabalhador ativo recebe em um determinado período é tal que a duração do período é a mesma para todos os processadores ativos neste período. Com esta nova restrição, os autores são capazes de derivar fórmulas analíticas fechadas para calcular o número de trabalhadores ativos n e o número de períodos m, bem como expressões analíticas para calcular o valor de  $\theta_i^j$  para todo  $i \in \{1, \ldots, n\}$  e  $j \in \{1, \ldots, m\}$ . Vale lembrar que  $\theta_i^j = 0$ , para todo  $i \in \{n + 1, \ldots, |P|\}$  e  $j \in \{1, \cdots, m\}$ . Experimentos computacionais mostraram que UMR superou os algoritmos de [11] e [56], em termos da qualidade da solução, que foram desenvolvidos para sistemas homogêneos e sistemas em único período, respectivamente.

Hsu et al. [37] propuseram uma abordagem heurística chamada *Extended Smallest Communication Ratio* - ESCR. A ordem de transmissão é definida da mesma maneira que em [67, 69]. No entanto, diferentemente de [67, 69], todos os processadores em P são ativados, ou seja, n = |P|. ESCR calcula o menor múltiplo comum (LCM) de  $\{G_i + w_i | i \in P\}$  e utiliza este valor para calcular a duração do período. O número de períodos m é calculado utilizando fórmulas analíticas fechadas baseadas em LCM e expressões analíticas são usadas para calcular o valor de  $\theta_i^j$  para todo  $i \in \{1, \ldots, n\}$  e  $j \in \{1, \ldots, m\}$ . Experimentos computacionais mostraram que ESCR supera o algoritmo de [5], desenvolvido para sistemas em único período.

Beaumont et al. [6] propuseram e analisaram quatro diferentes heurísticas para MR-DLS. A que obteve melhores resultados foi a heurística de programação linear com período adaptativo - LPAP. A ordem de transmissão é definida como em [67, 69]. Porém, a duração do período não é a mesma em cada período. Ao invés disso, LPAP calcula um limite T superior para o makespan. Com base nesta estimativa, a duração  $T_j$  do período j é definida como  $T_j = \sqrt{T}$ . Uma vez que a duração do período está definida, LPAP usa programação linear para encontrar o maior fragmento de carga  $\theta_i^j$  que pode ser enviado para cada processador  $i \in P$  no período j de modo que a duração do período não seja maior que  $T_j$ . Como o limite superior é mais apertado em cada iteração do algoritmo, a duração do período de um período j é sempre menor que a duração do período anterior. Experimentos computacionais mostram que LPAP supera o algoritmo de [11].

Shokripour et al. [60] propuseram duas heurísticas denominadas método baseado em computação e método baseado em comunicação. Os melhores resultados foram obtidos pelo último, referenciado aqui como CBM. Seja o tamanho do período a soma dos fragmentos de carga de todos os trabalhadores ativos em um dado período. CBM acrescenta uma restrição adicional de que todos os períodos têm o mesmo tamanho Q = W/m. Como nos trabalhos anteriores, a ordem de transmissão tem os trabalhadores ordenados em

ordem não-decrescente de  $G_i$  e os n primeiros processadores são ativados. Os valores de  $m \in n$  são calculados por fórmulas analíticas fechadas e expressões analíticas são usadas para calcular o valor de  $\theta_i^j$  para todo  $i \in \{1, \ldots, n\} \in j \in \{1, \ldots, m\}$ . Experimentos computacionais mostram que esta heurística obtém melhores resultados que as heurísticas LPAP [6] e ESCR [37]. Pode-se dizer que CBM era a heurística que apresentava os melhores resultados na literatura para MR-DLS até o desenvolvimento do algoritmo proposto nesta tese.

### 4.2 Contribuições

O estado da arte de algoritmos para MR-DLS foi levantado em [60], considerando uma ordem de transmissão fixa e fórmulas analíticas fechadas para decidir o número de trabalhadores ativos e o número de períodos, bem como expressões analíticas para calcular os tamanhos dos fragmentos de carga que cada trabalhador irá receber em cada período. Neste tese, foi proposta uma outra abordagem para o problema, onde um algoritmo genético com chaves aleatórias tendenciosas é utilizado.

Os resultados computacionais alcançados se encontram no artigo que compõe o Anexo D. Os resultados mostram que a heurística proposta, denominada GA-KEY, obteve resultados, em termos da qualidade da solução, 11,68% melhores do que CBM.

# Capítulo 5

## Conclusões

A principal contribuição desta tese foi mostrar o bom desempenho dos algoritmos genéticos com chaves aleatórias tendenciosas para diferentes problemas de otimização em redes.

A vantagem do BRKGA sobre o algoritmo genético clássico (AG) é o fato de sempre gerar soluções viáveis, desde que exista um algoritmo decodificador.

BRKGAs não tem bom desempenho quando a topografia do espaço de soluções é plana, como no caso do problema de *Steiner Triple Covering*. Neste caso, é preciso aumentar o número de mutantes e diminuir a probabilidade do descendente herdar a chave do pai elite. Com isso, o BRKGA fica parecido com o RKGA, o que não quer dizer que o BRKGA não funcione bem, mas sim que nesse tipo de problema ele não é significativamente melhor do que o RKGA.

Nesta tese, também foram propostas heurísticas construtivas para o problema de max-RWA que podem ser utilizadas como decodificadores para o BRKGA. Heurísticas mais elaboradas baseadas no BRKGA para os três problemas estudados foram propostas e os resultados mostraram que elas foram sempre melhores, em termos de qualidade da solução do que os métodos disponíveis na literatura para os respectivos problemas.

## Referências

- 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.
- [2] BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 2 (1994), 154–160.
- [3] BEAUMONT, O.; BONICHON, N.; EYRAUD-DUBOIS, L. Scheduling divisible workloads on heterogeneous platforms under bounded multi-port model. In *International Symposium on Parallel and Distributed Processing* (Miami, 2008), IEEE, pp. 1–7.
- [4] BEAUMONT, O.; CASANOVA, H.; LEGRAND, A.; ROBERT, Y.; YANG, Y. Scheduling divisible loads on star and tree networks: results and open problems. *IEEE Transactions on Parallel and Distributed Systems 16* (2005), 207–218.
- [5] BEAUMONT, O.; LEGRAND, A.; ROBERT, Y. The master-slave paradigm with heterogeneous processors. *IEEE Transactions on Parallel and Distributed Systems* 14 (2003), 897–908.
- [6] BEAUMONT, O.; LEGRAND, A.; ROBERT, Y. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. In 12th Heterogeneous Computing Workshop (Nice, 2003), IEEE Computer Society Press, pp. 98–111.
- [7] BELGACEM, L.; PUECH, N. Solving large size instances of the RWA problem using graph partitioning. In *International Conference on Optical Network Design and Modeling* (Barcelona, 2008), IEEE, pp. 1–6.
- [8] BERLINSKA, J.; DROZDOWSKI, M. Heuristics for multi-round divisible loads scheduling with limited memory. *Parallel Computing 36* (2010), 199–211.
- [9] BERLIŃSKA, J.; DROZDOWSKI, M.; LAWENDA, M. Experimental study of scheduling with memory constraints using hybrid methods. *Journal of Computational and Applied Mathematics 232* (2009), 638–654.
- [10] BHARADWAJ, V.; GHOSE, D.; MANI, V. Multi-installment load distribution in tree networks with delays. *IEEE Transactions on Aerospace and Electronic Systems 31* (1995), 555–567.
- [11] BHARADWAJ, V.; GHOSE, D.; MANI, V.; ROBERTAZZI, T. G. Scheduling divisible loads in parallel and distributed systems. Wiley - IEEE Computer Society Press, 1996.
- [12] BLAZEWICZ, J.; DROZDOWSKI, M. Distributed processing of divisible jobs with communication startup costs. Discrete Applied Mathematics 76 (1997), 21–41.
- [13] BRANDÃO, J. S.; NORONHA, T. F.; RESENDE, M. G. C.; RIBEIRO, C. C. A biased random-key genetic algorithm for scheduling divisible loads. In *Proceedings of* the Multidisciplinary International Scheduling Conference: Theory and Applications (Prague, 2015), pp. 570–578.
- [14] 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 Operational Research* 22 (2015), 823–839.
- [15] 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* (2015), 1–23.
- [16] BRANDÃO, J. S.; NORONHA, T. F.; RIBEIRO, C. C. A genetic algorithm for maximizing the accepted demands in routing and wavelength assignment in optical networks. In *Proceedings of the 11th Metaheuristics International Conference* (Agadir, 2015).
- [17] BURIOL, L. S.; HIRSCH, M. J.; PARDALOS, P. M.; QUERIDO, T.; RESENDE, M. G. C.; RITT, M. A biased random-key genetic algorithm for road congestion minimization. *Optimization Letters* 4 (2010), 619–633.
- [18] CHEN, C.; BANERJEE, S. A new model for optimal routing and wavelength assignment in wavelength division multiplexed optical networks. In Proceedings of the Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation (San Francisco, 1996), vol. 1, pp. 64–171.
- [19] CHOI, J. S.; GOLMIE, N.; LAPEYRERE, F.; MOUVEAUX, F.; SU, D. A functional classification of routing and wavelength assignment schemes in DWDM networks: Static case. In *Proceedings of the 7th International Conference on Optical Communication and Networks* (Paris, 2000), pp. 1109–1115.
- [20] DEBELS, D.; DE REYCK, B.; LEUS, R.; VANHOUCKE, M. A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. *European Journal of Operational Research 169* (2006), 638–653.
- [21] DROZDOWSKI, M. Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems. No. Monografia 321. Poznań University of Technology, 1997.
- [22] DROZDOWSKI, M.; LAWENDA, M. Multi-installment divisible load processing in heterogeneous systems with limited memory. *Lecture Notes in Computer Science* 3911 (2006), 847–854.
- [23] DROZDOWSKI, M.; LAWENDA, M. Multi-installment divisible load processing in heterogeneous distributed systems. *Concurrency and Computation: Practice and Experience 19* (2007), 2237–2253.
- [24] DROZDOWSKI, M.; WOLNIEWICZ, P. Divisible load scheduling in systems with limited memory. *Cluster Computing* 6 (2003), 19–29.
- [25] DROZDOWSKI, M.; WOLNIEWICZ, P. Optimum divisible load scheduling on heterogeneous stars with limited memory. *European Journal of Operational Research* 172 (2006), 545–559.

- [26] 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.
- [27] 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.
- [28] ERLEBACH, T.; JANSEN, K. The complexity of path coloring and call scheduling. *Theoretical Computer Science* 255 (2001), 33–50.
- [29] GONÇALVES, J. F.; DE MAGALHÃES MENDES, J. J.; RESENDE, M. G. C. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research* 167 (2005), 77–95.
- [30] GONÇALVES, J. F.; RESENDE, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics* 17 (2011), 487–525.
- [31] GONÇALVES, J. F.; RESENDE, M. G. C.; MENDES, J. J. A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *Journal of Heuristics* 17 (2011), 467–486.
- [32] 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.
- [33] GONÇALVES, J. F.; MENDES, J. J. M. A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics 8* (2002), 629–642.
- [34] GONÇALVES, J. F.; RESENDE, M. G. C. An evolutionary algorithm for manufacturing cell formation. Computers and Industrial Engineering 47 (2004), 247–273.
- [35] HOFFMANN, T. R. Eureka: A hybrid system for assembly line balancing. Management Science 38 (1992), 39–47.
- [36] HOLLAND, J. H. Adaptation in Natural and Artificial Systems. MIT Press, 1975.
- [37] HSU, C.-H.; CHEN, T.-L.; PARK, J.-H. On improving resource utilization and system throughput of master slave job scheduling in heterogeneous systems. *The Journal of Supercomputing* 45 (2008), 129–150.
- [38] JAUMARD, B.; MEYER, C.; THIONGANE, B. ILP formulations for the routing and wavelength assignment problem: Symmetric systems. In *Handbook of optimization* in telecommunications, M. G. C. Resende and P. M. Pardalos, Eds. Springer, 2006, pp. 637–677.
- [39] JAUMARD, B.; MEYER, C.; THIONGANE, B. Comparison of ILP formulations for the RWA problem. Optical Switching and Networking 4 (2007), 157–172.
- [40] JAUMARD, B.; MEYER, C.; THIONGANE, B. On column generation formulations for the RWA problem. Discrete Applied Mathematics 157 (2009), 1291–1308.

- [41] JAUMARD, B.; MEYER, C.; THIONGANE, B.; YU, X. ILP formulations and optimal solutions for the RWA problem. In *Proceedings of the IEEE Global Telecommunications Conference* (Dallas, 2004), vol. 3, pp. 1918–1924.
- [42] KIM, H. J. A novel optimal load distribution algorithm for divisible loads. Cluster Computing- The Journal of Networks Software Tools and Applications 6 (2003), 41– 46.
- [43] KLEINBERG, J. Approximation algorithms for disjoint paths problems. Tese de Doutorado, MIT, Cambridge, 1996.
- [44] KRISHNASWAMY, R.; SIVARAJAN, K. Algorithms for routing and wavelength assignment based on solutions of LP-relaxation. *IEEE Communications Letters* 5 (2001), 435–437.
- [45] LEE, T.; LEE, K.; PARK, S. Optimal routing and wavelength assignment in wdm ring networks. *IEEE Journal on Selected Areas in Communications* 18 (2000), 2146– 2154.
- [46] LI, X.; BHARADWAJ, V.; KO, C. Divisible load scheduling on single-level tree networks with buffer constraints. *IEEE Transactions on Aerospace and Electronic Systems 36* (2000), 1298–1308.
- [47] LIN, X.; DEOGUN, J.; LU, Y.; GODDARD, S. Multi-round real-time divisible load scheduling for clusters. *Lecture Notes in Computer Science* 5374 (2008), 196–207.
- [48] M. GROTSCHEL, L. LOVÁSZ, A. S. Geometric Algorithms and Combinatorial Optimization, vol. 2. Algorithms and Combinatorics, Springer-Verlag, 1993.
- [49] MACIEJ, D.; MARCIN, L. Scheduling multiple divisible loads in homogeneous star systems. *Journal of Scheduling 11* (2008), 347–356.
- [50] MANOHAR, P.; MANJUNATH, D.; SHEVGAONKAR, R. K. Routing and wavelength assignment in optical networks from edge disjoint path algorithms. *IEEE Commu*nications Letters 5 (2002), 211–213.
- [51] 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, 2011.
- [52] MARTINS, A. X.; DUHAMEL, C.; MAHEY, P.; DE SOUZA, M. C.; SALDANHA, R. R. Geração de colunas para o problema de roteamento e atribuição de comprimentos de onda. In Anais do XLIV Simpósio Brasileiro de Pesquisa Operacional (Rio de Janeiro, 2012), pp. 3268–3279.
- [53] NORONHA, T. F.; 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.
- [54] NORONHA, T. F.; RIBEIRO, C. C. Routing and wavelength assignment by partition coloring. European Journal of Operational Research 171 (2006), 797–810.

- [55] RAMASWAMI, R.; SIVARAJAN, K. Optimal routing and wavelength assignment in all-optical networks. In *Proceedings of IEEE INFOCOM Conference on Computer Communications* (Toronto, 1994), pp. 970–979.
- [56] ROSENBERG, A. L. Sharing partitionable workloads in heterogeneous nows: greedier is not better. In *cluster* (2001), IEEE, p. 124.
- [57] SCHOLL, A.; VOSS, S. Simple assembly line balancing heuristic approaches. Journal of Heuristics 2 (1997), 217–244.
- [58] SHOKRIPOUR, A.; OTHMAN, M.; IBRAHIM, H. A method for scheduling last installment in a heterogeneous multi-installment system. In *Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology* (Chengdu, 2010), pp. 714–718.
- [59] SHOKRIPOUR, A.; OTHMAN, M.; IBRAHIM, H.; SUBRAMANIAM, S. A new method for job scheduling in a non-dedicated heterogeneous system. *Procedia Computer Science* 3 (2011), 271–275.
- [60] SHOKRIPOUR, A.; OTHMAN, M.; IBRAHIM, H.; SUBRAMANIAM, S. New method for scheduling heterogeneous multi-installment systems. *Future Generation Computer* Systems 28 (2012), 1205–1216.
- [61] SKORIN-KAPOV, N. Routing and wavelength assignment in optical networks using bin packing based algorithms. *European Journal of Operational Research* 177 (2007), 1167–1179.
- [62] SPEARS, W.; DEJONG, 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] WANG, M.; WANG, X.; MENG, K.; WANG, Y. New model and genetic algorithm for divisible load scheduling in heterogeneous distributed systems. *International Journal* of Pattern Recognition and Artificial Intelligence 27 (2013), 1359005-1-1359005-18. Referência online em: http://www.worldscientific.com/doi/abs/10.1142/ S0218001413590052?journalCode=ijprai, último acesso em 16/11/2015.
- [64] WANG, X.; WANG, Y.; MENG, K. Optimization algorithm for divisible load scheduling on heterogeneous star networks. *Journal of Software 9* (2014), 1757–1766.
- [65] WOLNIEWICZ, P. Divisible Job Scheduling in Systems with Limited Memory. Tese de Doutorado, Poznan University of Technology, Poznań, 2003.
- [66] YANG, Y.; CASANOVA, H. RUMR: Robust scheduling for divisible workloads. In Proceedings of the 12th IEEE Symposium on High Performance and Distributed Computing (Seattle, 2003), pp. 114–125.
- [67] YANG, Y.; CASANOVA, H. UMR: A multi-round algorithm for scheduling divisible workloads. In Proceedings of the 17th International Parallel and Distributed Processing Symposium (Nice, 2003), pp. 24–32.

- [68] YANG, Y.; CASANOVA, H.; DROZDOWSKI, M.; LAWENDA, M.; LEGRAND, A., ET AL. On the complexity of multi-round divisible load scheduling. Tech. Rep. 6096, INRIA Rhône-Alpes, 2007.
- [69] YANG, Y.; VAN DER RAADT, K.; CASANOVA, H. Multi-round algorithms scheduling divisible loads. *IEEE Transactions on Parallel and Distributed Systems 16* (2005), 1092–1102.
- [70] ZANG, H.; JUE, J. P.; MUKHERJEE, B. A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks. *Optical Networks Magazine* 1 (2000), 47–60.

ANEXO A – 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 (2015), p. 1 - 23. **Journal of Global Optimization manuscript No.** (will be inserted by the editor)

# A biased random-key genetic algorithm to maximize the number of accepted lightpaths in WDM optical networks

Julliany S. Brandão<sup>1</sup>, Thiago F. Noronha<sup>2</sup>, Celso C. Ribeiro<sup>1</sup>

Received: date / Accepted: date

Abstract Given a set of lightpath requests, the problem of routing and wavelength (RWA) assignment in WDM optical networks consists in routing a subset of these requests and assigning a wavelength to each of them, such that two lightpaths that share a common link are assigned to different wavelengths. There are many variants of this problem in the literature. We focus in the variant in which the objective is to maximize the number of requests that may be accepted, given a limited set of available wavelengths. This problem is called max-RWA and it is of practical and theoretical interest, because algorithms for this variant can be extended for other RWA problems that arise from the design of WDM optical networks. A number of exact algorithms based on integer programming formulations have been proposed in the literature to solve max-RWA, as well as algorithms to provide upper bounds to the optimal solution value. However, the algorithms based on the state-of-the-art formulations in the literature cannot solve the largest instances to optimality. For these instances, only upper bounds to the value of the optimal solutions are known. The literature on heuristics for max-RWA is short and focus mainly on solving small size instances with up to 27 nodes. In this paper, we propose new greedy constructive heuristics and a biased random-key genetic algorithm (BRKGA), based on the best of the proposed greedy heuristics. Computational experiments showed that the new heuristic outperforms the best ones in literature. Furthermore, for the largest instances in the literature where only upper bounds to the value of the optimal solutions are known, the average optimality gap of the best of the proposed heuristics is smaller than 4%.

**Keywords** Random-key Genetic Algorithms · Metaheuristics · Applications · Optical Networks · Routing and Wavelength Assignment

<sup>1</sup> Universidade Federal Fluminense (UFF),

This work was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq), the Foundation for Support of Research of the State of Minas Gerais (FAPEMIG), the Foundation for Support of Research of the State of Rio de Janeiro (FAPERJ), and the Coordination for the Improvement of Higher Education Personnel, Brazil (CAPES).

Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil

E-mail: {jbrandao, celso}ic.uff.br

<sup>&</sup>lt;sup>2</sup> Universidade Federal de Minas Gerais (UFMG),

Avenida Antônio Carlos 6627, Belo Horizonte, MG 31270-901, Brazil. E-mail: tfn@dcc.ufmg.br

#### 1 Introduction

In optical networks, information is transmitted as optical signals through optical fibers. Each optical link operates at a speed of the order of terabits per second, which is much faster than the currently available optical devices. *Wavelength Division Multiplexing* (WDM) allows the more efficient use of the capacity of the optical fibers, as far as it permits the simultaneous transmission of different signals along the same fiber, provided they are multiplexed with different wavelengths. An all-optical point-to-point connection between two nodes is called a *lightpath*. Each lightpath is characterized by its route and by the wavelength with which it is multiplexed. Two lightpaths may use the same wavelength, provided they do not share any common fiber.

Given the physical topology of an optical network and a set of lightpaths defining a logical topology in this network, the problem of Routing and Wavelength Assignment (RWA) in WDM optical networks consists in routing the set of lightpaths and assigning a wavelength to each of them, such that lightpaths whose routes share a common fiber are assigned to different wavelengths. Variants of RWA are characterized by different optimization criteria and traffic patterns, see e.g. [12,57]. We consider RWA variants in which the lightpath requests are known beforehand and no wavelength conversion is available, i.e. a lightpath must be assigned to the same wavelength on all fibers along its route. Let G = (V,A) be a directed graph representing the physical network topology, where V is the set of the nodes and A represents the fiber connections between the nodes. Let also R be the set of lightpaths requests, each one defined by a source and a destination node in V. There can be more than one lightpath request between any pair of nodes, since the traffic between a pair of nodes can be larger than that supported by a single lightpath. We denote by  $\lambda$  the number of available wavelengths. In the min-RWA problem variant [15], the number of available wavelengths is unbounded and the objective is to minimize the number of wavelengths used to establish all lightpath requests in R. This paper focus on the max-RWA variant [11], where  $\lambda < |R|$ . Therefore, it may not be possible to accept and route all requests in R. The objective is to maximize the number of requests that may be accepted. Both problems have been proved to be NP-hard in [15] and [11], respectively.

Exact algorithms based on integer programming formulations have been proposed in the literature. However, as the worst case complexity of these algorithms grows exponentially with the size of the network, they can only solve small instances to optimality. State-of-the-art exact algorithms can only provide upper bounds to the optimal value for the largest instances in the literature. To the best of our knowledge, the only existing heuristics for max-RWA are the greedy algorithms in [34,41], the graph partitioning approach in [4], and the tabu search heuristic in [13].

In this paper, we propose heuristics for efficiently solving large instances in the literature. Related work is reviewed in Section 2. Greedy constructive heuristics are proposed in Section 3. A biased random-key genetic algorithm (BRKGA), based on the best of the greedy heuristics, is proposed in Section 4. Computational experiments are reported in Section 5, where it is shown that the proposed heuristics outperform the previous heuristics in the literature. Finally, concluding remarks are drawn in the last section.

#### 2 Related work

#### 2.1 Exact formulations and algorithms for max-RWA

Most of the literature on algorithms for solving max-RWA reports on integer programming formulations and exact approaches [29,30,34,35,34,46,50]. A review on formulations for max-RWA can be found in [27,28,43].

Krishnaswamy and Sivarajan [34] developed an arc based compact formulation for max-RWA. This formulation allows cycles, but they argue that these cycles have no impact in the value of the objective function. A cycle free compact formulation based on the previous one was proposed in [30]. This formulation does not improve the upper bounds of the former, but it can be solved more efficiently by MIP solvers. Martins [43] improved the compact formulations of [30, 34] and found optimal solutions for instances with up to 18 nodes.

Lee et al. [35] proposed a formulation based on the weighted independent set problem with additional cardinality constraints [39], whose linear relaxation can be solved by a column generation algorithm. Jaumard et al. [29,30] proposed an improved formulation that uses only maximal independent sets and found better upper bounds than those of [35] for instances with up to 27 nodes.

Ramaswami and Sivarajan [50] proposed a maximum independent set formulation with an exponential number of variables. They proved that the upper bound provided by its linear relaxation is never smaller than that of the best compact integer programming formulations found in the literature at the time of its publication. Jaumard et al. [29] implemented a column generation algorithm based on this formulation and solved instances with up to 27 nodes to optimality.

Martins et al. [43,46] proposed a new, improved formulation based on those in [29,35, 50]. Their approach obtained the first upper bounds for the three instances based on the 71node ATT2 network, that could not be computed by previous works in the literature due to memory overflow. They found the best upper bounds for the largest instances in the literature of max-RWA with up to 90 nodes. However, not even feasible solutions have been reported for these instances.

#### 2.2 Heuristics for max-RWA

To the best of our knowledge, there are few heuristics for max-RWA in the literature. Krishnaswamy and Sivarajan [34] proposed two rounding heuristics based on the linear relaxation of their integer programming formulations. Computational experiments carried out on two networks with 14 and 20 nodes showed that the average relative optimality gap for the best of the two heuristics was of 6.0% and 7.2%, respectively.

Manohar, Manjunath, and Shevgaonkar [41] developed the Greedy-EDP-RWA heuristic. At each iteration, a subset of lightpaths is selected and routed with edge disjoint paths by the BGAforEDP heuristic for the maximum edge disjoint path (EDP) problem [33]. Then, all lightpaths in this subset are assigned the same wavelength, and the procedure is repeated with the remaining lightpaths. This heuristic was proposed for the min-RWA variant of the problem, but the authors argue that it can also be used for max-RWA, by running BGAforEDP for  $\lambda$  iterations. The authors reported that their algorithm was faster than and found solutions as good as the linear programming based algorithms for min-RWA at the time of its publication. No computational experiments on the performance of this heuristic for max-RWA have been reported by the authors.

A tabu search heuristic was proposed by Dzongang et al. [13]. First, this heuristic builds a set  $P_r$  of pre-computed paths between the endnodes of each request  $r \in R$ . Each neighbor of the current solution *S* is generated by using a path  $p \in P_r$  and a wavelength  $\omega \in \{1, ..., \lambda\}$ to route a request *r* not accepted in the solution *S* and then removing from *S* all requests assigned to wavelength  $\omega$  that share a link with *p*. Computational experiments on the same instances used in [34] have shown that tabu search found much better solutions than the rounding heuristics of [34], with average relative optimality gaps of 1.41% and 1.53% for the instances with 14 and 20 nodes, respectively.

Belgacem and Puech [4] proposed a decomposition heuristic for max-RWA, in which the original instance is partitioned into smaller instances which are exactly solved by integer programming. The local solutions are combined into a feasible solution. This proposal was validated by an application to the European backbone network EBN57 with 57 nodes and to randomly generated planar networks with up to 500 nodes.

#### 2.3 Heuristics for variants of max-RWA with wavelength conversion

Marković et al. [42] proposed a bee colony optimization heuristic for the variant of max-RWA in which the wavelength conversion is available at some nodes. Computational experiments showed that the heuristic was able to produce optimal or near-optimal solutions only for two small networks with 8 and 18 nodes.

Qin et al. [49] proposed a new optimization objective, which consists in determining the maximum number of connections with the least number of wavelength converters. The problem was tackled by a genetic algorithm.

Jaumard et al. [31] proposed a new two-phase heuristic for the variant in which wavelength conversion is possible in every node. Wavelength assignment is reformulated as a generalized partition coloring problem, extending the partition coloring formulation proposed by Noronha and Ribeiro [16]. This problem was solved by a tabu search heuristic. Computational experiments on instances with up to 27 nodes have shown that wavelength conversion is of little help to increase the number of accepted lightpaths, except for some very particular traffic patterns [31].

#### 2.4 Heuristics for min-RWA

Contrarily to the max-RWA variant considered in this paper, most of the research on algorithms for solving the min-RWA variant is focused into heuristics. Some approaches decompose the problem into two subproblems (routing and wavelength assignment) [2,25,26,37, 16], while others tackle the two subproblems simultaneously [41,43–46,48,54].

Skorin-Kapov [54] proposed the current state-of-the-art greedy constructive heuristics for min-RWA. Each wavelength is associated with a different copy of *G*. Lightpaths that are routed along disjoint arcs on the same copy of *G* are assigned to the same wavelength. Copies of *G* are associated with the bins and lightpaths with the items of an instance of the *bin packing problem* [32]. In this context, min-RWA can be reformulated as the problem of packing all the lightpath requests (items) in a minimum number of wavelength (bins). Four min-RWA heuristics based on classical bin packing heuristics were developed: (i) FF-RWA, based on the *first fit heuristic*, (ii) BF-RWA, based on the *best fit heuristic*, (iii) FFD-RWA, based on the *first fit decreasing* heuristic, and (iv) BFD-RWA, based on the *best fit decreasing* heuristic.

Computational results have shown that the best results have been obtained by BFD-RWA, whose pseudo-code is presented in Figure 1. The inputs are the graph *G*, the set *R* of lightpath requests, the value *d* of the maximum number of arcs allowed in each route, and the vector  $\pi = (\pi(1), ..., \pi(|R|))$  describing the order in which the lightpaths will be considered. Let *min-length*(*i*) be the number of hops in the path in *G* with the smallest number of arcs between the endnodes of lightpath  $i \in R$ . The lightpaths will be considered in a non-increasing order of their *min-length* values, i.e.,  $\pi(k) = \operatorname{argmax}\{\min-\operatorname{length}(i) : i \in R \setminus \{\pi(1), ..., \pi(k-1)\}\}$ , for any k = 1, ..., |R|. Ties between two lightpaths with the same min-length value are broken arbitrarily. The idea behind this ordering is that long lightpaths are harder to be routed and therefore should be routed first. The output is a set *S* of pairs in which the first element is the path used to route lightpath *i* and the second is the wavelength with which it is multiplexed.

Each solution is formed by a set of pairs, each of them containing the route and the wavelength assigned to one lightpath. The solution *S* and the number *k* of lightpaths used are initialized in line 1. The lightpaths are routed one at a time (and assigned to a wavelength) in lines 2 to 14. In line 3, the algorithm determines whether lightpath  $\pi(i)$  can be routed using any of the *k* wavelengths already used. If this is the case, then in line 5 the algorithm determines wavelength *r* as that in which lightpath  $\pi(i)$  can be routed with the least number of arcs. Otherwise, the number of wavelengths used is increased by one in line 7, a new copy of graph *G* is created in line 8, and the new wavelength r = k is selected to be assigned to lightpath  $\pi(i)$  in line 9. Line 11 computes the shortest path  $P_{\pi(i)}$  between the endnodes of lightpath  $\pi(i)$  in *G*<sub>r</sub>. In line 12, the pair  $(P_{\pi(i)}, r)$  is added to the solution under construction and all arcs in route  $P_{\pi(i)}$  used by lightpath  $\pi(i)$  are removed from *G*<sub>r</sub> in line 13. A feasible solution *S* and the number *k* of wavelengths used are returned in line 15.

```
procedure BFD-RWA(G, R, d, \pi)
      Set S \leftarrow \emptyset, \Omega \leftarrow \emptyset, and k \leftarrow 0:
1
      for i = 1, ..., |R| do
2
            if there is a path for routing lightpath \pi(i) with less than d arcs in any of the graphs G_1, \ldots, G_k
3
4
            then
5
                   Let r \in \{1, ..., k\} such that lightpath \pi(i) can be routed in graph G_r with the
                   least number of arcs:
6
            else
                   k \leftarrow k+1;
7
8
                   G_k \leftarrow G;
9
                   r \leftarrow k;
10
            end-if:
11
            Let P_{\pi(i)} be the shortest path between the endnodes of lightpath \pi(i) in G_r;
12
            S \leftarrow S \cup \{(p_{\pi(i)}, r)\};
13
             Remove all arcs in path P_{\pi(i)} from G_r;
14
      end-for:
15
      return S,k;
end BFD-RWA
```

Fig. 1 Pseudo-code of the BFD-RWA heuristic for min-RWA.

Noronha et al. [47] developed efficient algorithms and data structures for the implementation of heuristics FF-RWA, BF-RWA, FFD-RWA, and BFD-RWA. The longest running times of their implementation of BFD-RWA took less than three seconds, while the times reported for the heuristic in [54] took up to eight minutes on the same instances and the same Pentium IV 2.8 GHz processor. BFD-RWA was successfully used in the iterated local search heuristic of Martins at al. [45] and in the biased random-key genetic algorithm of Noronha et al. [48] for the problem min-RWA. Computational experiments have shown that these are to date the best heuristics for min-RWA. These results motivated the development of constructive heuristics based on BFD-RWA and the development of a biased random-key genetic algorithm for the max-RWA problem variant.

#### 3 Greedy constructive heuristics for max-RWA

State-of-the-art exact algorithms and heuristics for max-RWA perform well only for small instances with 27 nodes. In Section 3.1, we present three greedy heuristics for max-RWA based on bin packing that are extensions of the best heuristic (BFD-RWA) for min-RWA. Next, we propose in Section 3.2 three new greedy heuristics for max-RWA that are based on multi-processor scheduling. Later, we will show that the heuristics contribute effectively in the solution of the largest instances in the literature of max-RWA with up to 90 nodes.

# 3.1 Constructive heuristics based on bin packing

We recall that for min-RWA one may use as many wavelengths as necessary, while for max-RWA there are at most  $\lambda$  available wavelengths for routing. The three heuristics (BFR-RWA, BFI-RWA, and BFD-RWA) proposed in this section are derived from BFD-RWA and differ by the order in which the lightpaths are considered:

- BFR: requests are randomly selected from *R*;
- BFI: requests are sorted in non-decreasing order of their min-length values; and
- BFD: as for BFD-RWA, requests are sorted in non-increasing order of their min-length values, with ties arbitrarily broken.

A general framework BF\* for the pseudo-code of the family of heuristics BFR, BFI, and BFD is given in Figure 2. The inputs are the graph G, the set R of lightpath requests, the maximum number d of arcs in each route, a permutation  $\pi(1), \ldots, \pi(|R|)$  describing the order in which the requests are considered, and the maximum number  $\lambda$  of wavelengths available. As for min-RWA, each solution is formed by a set of pairs, each of them containing the route and the wavelength assigned to one lightpath. The solution S and the number k of lightpaths used are initialized in line 1. The lightpaths are routed one at a time (and assigned to a wavelength) in lines 2 to 14. In line 3, sets r to zero as a flag in case lightpath  $\pi(i)$ can not be the routed with the  $\lambda$  available wavelengths. The algorithm determines in line 4 whether lightpath  $\pi(i)$  can be routed using any of the k wavelengths already used. If this is the case, then in line 6 the algorithm determines wavelength r as that in which lightpath  $\pi(i)$  can be routed with the least number of arcs. Otherwise, and if not all wavelengths have already been used, the number of wavelengths used is increased by one in line 8, a new copy of graph G is created in line 9, and the new wavelength r = k is selected to be assigned to lightpath  $\pi(i)$  in line 10. If lightpath  $\pi(i)$  can be routed, then line 14 computes the shortest path  $P_{\pi(i)}$  between the endnodes of lightpath  $\pi(i)$  in  $G_r$ . In line 15, the pair  $(P_{\pi(i)}, r)$  is added to the solution under construction and all arcs in route  $P_{\pi(i)}$  used by lightpath  $\pi(i)$  are removed from  $G_r$  in line 16. A feasible solution S and the number k of wavelengths used are returned in line 19.

6

A BRKGA for max-RWA in WDM optical networks

```
procedure BF*(G, R, d, \pi, \lambda)
      Set S \leftarrow \emptyset and k \leftarrow 0;
      for i = 1, ..., |R| do
2
3
            r \leftarrow 0;
            if there is a path for routing lightpath \pi(i) with less than d arcs in any of the graphs G_1, \ldots, G_k
4
5
            then
                  Let r \in \{1, ..., k\} such that lightpath \pi(i) can be routed in graph G_r with the
6
                  least number of arcs:
7
            else if k < \lambda then
8
                        k \leftarrow k+1:
9
                        G_k \leftarrow G;
10
                        r \leftarrow k;
                  end-if;
11
            end-if;
12
13
            if r > 0 then
                  Let P_{\pi(i)} be the shortest path between the endnodes of lightpath \pi(i) in G_r;
14
15
                  S \leftarrow S \cup \{(P_{\pi(i)}, r)\};
                  Remove all arcs in path P_{\pi(i)} from G_r;
16
17
            end-if
18
      end-for:
19
      return S,k;
end BF*.
```

Fig. 2 Pseudo-code of the family BF\* of constructive heuristics for max-RWA based on bin packing.

#### 3.2 Constructive heuristics based on multi-processor scheduling

In this section, we propose three new heuristics for max-RWA based on a classic heuristic for the *Multi-Processor Scheduling Problem* (MPSP) [22]. Given a set *P* of processors and a set *T* of tasks, where each task  $t \in T$  is associated with a running time  $b_t$ , the heuristic consists in assigning a processor  $p_t \in P$  to each task  $t \in T$ , in order to minimize the *maximum completion time* of a processor. If  $T_p \subseteq T$  denotes the subset of tasks assigned to each processor  $p \in P$  in a given solution, the maximum completion time of this solution is given by  $\max_{p \in P} \{\sum_{t \in T_p} b_t\}$ .

The *Longest Processing Time* heuristic [24] builds a feasible solution for MPSP as follows. First, it sorts the tasks in non-increasing order of their completing time. Following this order, each task is scheduled to one of the available processors in which the maximum completing time increases the least.

The three heuristics for max-RWA described below are inspired by this algorithm. Each wavelength is associated with a different copy of G. Lightpaths that are routed along disjoint arcs on the same copy of G are assigned the same wavelength. Each copy of G is associated with one processor and each lightpaths with one of the tasks of an instance of MPSP. The completion time of each task is defined as the min-length of the corresponding lightpath. Therefore, max-RWA can be reformulated as the problem of scheduling the tasks (lightpath requests) in the set of processors (wavelengths).

The pseudo-code of the new family \*PT of heuristics is presented in Figure 3. As for BFR, BFI and BFD, the inputs are the graph *G*, the maximum number *d* of arcs in each route, the set *R* of lightpath requests, a permutation  $\pi(1), \ldots, \pi(|R|)$  describing the order in which the lightpaths are considered, and the maximum number  $\lambda$  of wavelengths available. Once again, the three heuristics (RPT, SPT, and LPT) differ by the order in which the lightpaths are considered:

- RPT: requests are randomly selected from *R*;
- SPT: requests are sorted in non-decreasing order of their min-length values; and
- LPT: as for the Longest Processing Time heuristic, requests are sorted in non-increasing order of their min-length values, with ties arbitrarily broken.

Algorithms \*PT in Figure 3 are very similar to those named as BF\* in Figure 2. Basically, the only difference consists that in the family \*PT all processors are available (or, in other words, all copies of the original graph are built beforehand) since the beginning of the algorithm. As the numerical results will show, the underlying strategy of \*PT is better, since creating all copies of the graphs beforehand makes it possible to assign the best route (i.e., that with the least number of edges) among all copies of the graph to each lightpath. Consequently, the edges blocked to be used for each lightpath will be the least possible and will leave more choices for the lightpaths that will be considered next.

```
procedure *PT(G, R, d, \pi, \lambda)
      Set S \leftarrow \emptyset and k \leftarrow 0;
2
      for k = 1, \ldots, \lambda do
3
            G_k \leftarrow G;
4
      end-for;
5
      for i = 1, ..., |R| do
6
            if lightpath \pi(i) can be routed with less than d arcs in any of the graphs G_1, \ldots, G_{\lambda}
7
            then
8
                  k \leftarrow k+1;
9
                  Let r \in \{1, ..., \lambda\} such that lightpath \pi(i) can be routed in graph G_r with the
                   least number of arcs;
10
                   Let P_{\pi(i)} be the shortest path between the endnodes of lightpath \pi(i) in G_r;
11
                  S \leftarrow S \cup \{(P_{\pi(i)}, r)\};
                  Remove all arcs in path P_{\pi(i)} from G_r;
12
13
            end-if
      end-for
14
15
      return S.k:
end *PT.
```

Fig. 3 Pseudo-code of the family \*PT of constructive heuristics for max-RWA based on multi-processor scheduling.

Heuristics BFR, BFI, BFD, RPT, SPT, LPT, and GREEDY-EDP-RWA [41] will be evaluated and compared in Section 5.

#### 4 Biased random-key genetic algorithm

Genetic algorithms with random keys, or random-key genetic algorithms (RKGA), were first introduced by Bean [3] for combinatorial optimization problems for which solutions can be represented as a permutation vector. 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 a RKGA. Parents are allowed to be selected for mating more than once in a given generation.

A biased random-key genetic algorithm (BRKGA) differs from a RKGA in the way parents are selected for crossover, see Gonçalves and Resende [18] 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. We say the selection is biased since one parent is always an elite individual and because this elite solution has a higher probability of passing its genes to the offsprings, i.e. to the new generation. A BRKGA provides a better implementation of the essence of Darwin's principle of "survival of the fittest", since an elite solution has a higher probability of being selected for mating and the offsprings have a higher probability of inheriting the genes of the elite parent.

The development and application of a BRKGA to max-RWA was motivated by successful applications to many network optimization problems, such as other variants of routing and wavelength assignment [21,47,48,16], routing in OSPF networks [7,8,14,51], road congestion [6], as well as to other combinatorial optimization problems, such as job shop scheduling [17,36], assembly line balancing [19], lateness minimization on parallel batch processing machines [40,56], and cell formation in manufacturing [20], among others.

The biased random-key genetic algorithm for max-RWA evolves a population of chromosomes that consists of vectors of real numbers (called keys). Each solution is represented by an |R|-vector, in which each component is a real number in the range [0, 1] associated with a lightpath request in R. Each solution represented by a chromosome is decoded by a decoding heuristic that receives the vector of keys and builds a feasible solution for max-RWA.

Any of the heuristics BF\* or \*PT presented in Section 3 can be used as a decoding heuristic for BRKGA. Since the computational experiments that will be presented in the next section have shown that SPT obtains the best results among the seven tested greedy constructive heuristics, the description of BRKGA that follows will consider SPT as the decoder. The decoding consists of two steps. First, the lightpaths are sorted in non-decreasing order of the sum of their min-length and key values. Therefore, the relative order between lightpaths with the same min-length value is defined by their keys. The resulting order is used as the vector  $\pi$  in SPT (see the pseudo-code in Figure 3). The number of wavelengths found by SPT using this order is used as the fitness of the chromosome. The algorithm stops when a maximum elapsed time is reached or when a solution as good as a given target is found.

We use the parametrized uniform crossover scheme proposed in [55] to combine two parent solutions and produce an offspring. In this scheme, the offspring inherits each of its keys from the best fit of the two parents with probability  $\rho > 0.5$  and from the least fit parent with probability  $1 - \rho$ . This genetic algorithm 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: a fixed number of 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.

The keys associated to each lightpath request are randomly generated in the initial population. At each generation, the population is partitioned into two sets: *TOP* and *REST*. Consequently, 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 on the current population. As illustrated in Figure 4, 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



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

with one parent randomly chosen from *TOP* and the other from *REST*. This distinguishes a biased random-key GA from the random-key genetic algorithm of Bean [3] (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** Computational experiments

The heuristics Greedy-EDP-RWA[41], BFD, BFR, BFI, LPT, RPT, SPT, and BRKGA were implemented in C++ and compiled with GNU C++ version 4.6.3. The inheritance probability  $\rho$  of the crossover operator in BRKGA was set to 0.7, as used and recommended in [5, 17,21,48,52,58]. The population size was set to |TOP| + |MID| + |BOT| = |V|, with the sizes of sets *TOP*, *REST*, and *BOT* set to  $0.25 \times |V|$ ,  $0.7 \times |V|$ , and  $0.05 \times |V|$ , respectively, once again as suggested and used in [21,48] for the problem of routing and wavelength assignment. The experiments were performed on a 3.40 GHz i7-4770 Intel Core CPU with 16 GB of RAM memory.

Four sets of instances have been used in the experiments. The physical topologies are connected and each link corresponds to a pair of bidirectional fibers. The logical topology is asymmetric, i.e. there might be a lightpath request from a node *i* to a node *j*, while not from *j* to *i*. The first three sets were proposed in [46] and have 24 instances each. They are based in the same 24 networks with up to 90 nodes, but differ from each other by the number of wavelengths available. Set A is formed by instances in which there are ten wavelengths available. Instances in sets B and C have 20 and 30 wavelengths available, respectively. These are the largest and most difficult instances in the literature of max-RWA. No feasible solutions are available in the literature, only the upper bounds obtained by the column generation algorithm of [43]. The main characteristics of the networks used in sets A, B, and C are presented in Table 1, which gives respectively, the network name, the number of lightpath requests, the number of nodes, and the number of arcs of each network.

The fourth set of instances, referred here as D, was used by Dzongang et al. [13] to evaluate the performance of their tabu search heuristic, and to show that the latter outperforms the two rounding heuristics of [34]. It has 30 instances based on the NSF and EONNET networks. These networks have the same physical topologies of the networks NSF1 and EON of Table 1, respectively, but have different sets of lightpath requests and different values for  $\lambda$ . NSF has 14 nodes, 21 links, and 268 lightpath requests, while EONNET has 20 nodes, 39 links, and 374 lightpath requests. Fifteen instances are based on NFS and have from 10 to 24 wavelengths available, while the other fifteen instances are based on EONNET and also have from 10 to 24 wavelengths available.

Name	Lightpaths	Nodes	Links
ATT	359	90	137
ATT2	2918	71	175
BRASIL	1370	27	70
COST266	6543	37	57
DFN-BWIN	4840	10	45
DFN-GWIN	3771	11	47
EON	373	20	39
FINLAND	930	31	51
FRANCE	15398	25	45
GIUL	14732	39	86
JANOS-US	3262	26	42
NOBEL-EU	1898	28	41
NOBEL-GERMANY	660	17	26
NOBEL-US	478	14	21
NORWAY	5348	27	51
NSF1	284	14	21
NSF12	551	14	21
NSF21	284	14	22
NSF212	551	14	22
NSF23	285	14	22
NSF248	547	14	22
NSF3	285	14	21
NSF48	547	14	21
SUN	952	27	51

 Table 1 Main characteristics of the networks in sets A, B, and C.

In the first experiment, we evaluate and compare the quality of the solutions provided by the heuristics Greedy-EDP-RWA, BFD, BFR, BFI, LPT, RPT, and SPT for the instances in sets A, B, and C. The vectors  $\pi$  describing the order in which the lightpaths are considered by heuristics Greedy-EDP-RWA, BFR, and RPT have been randomly generated. Ties between two lightpaths with the same min-length value were broken randomly for heuristics BFD, BFI, LPT, and SPT. Each heuristic was run 10 times for each instance, with different seeds for the random number generator [53]. The average optimality gap (UB – LB)/UB between the value LB of the solution provided by the heuristic and the upper bound UB presented in [43] is calculated for each instance.

The average gap over all instances in each set is displayed in Figure 5 for each constructive heuristic. We first observe that the relative behavior of the seven heuristics is absolutely the same for the three test sets. We notice that BFD and LPT led to the largest average gaps among the seven tested heuristics. This is due to the fact that in these algorithms the lightpaths requests are sorted in non-increasing order of their min-length values, that is, the hardest lightpath to route is the first inserted into the solution. Although this greedy criterion often leads to good results for other problems, in situations such as max-RWA it might be better not to accept long lightpaths (i.e., those with large min-length values) in order to



Fig. 5 Average relative optimality gaps for heuristics Greedy-EDP-RWA[41], BFD, BFR, BFI, LPT, RPT, SPT for instance sets A, B, and C.

save space for shorter lightpaths (i.e., with smaller min-length values). The best results have been obtained by heuristics BFI and SPT for all test sets, with both of them considering the lightpath requests in a non-decreasing order of their min-length values. The average optimality gaps observed for BFI and SPT were, respectively, 13.36% and 9.41% for set A, 11.21% and 8.07% for set B, and 8.24% and 5.73% for set C. Heuristic SPT proposed in this work outperformed the BFI heuristic based on the work of [54]. In addition, the average optimality gaps provided by SPT were less than half of those observed with the Greedy-EDP-RWA [41] heuristic, which amounted to 30.94%, 22.13%, and 16.53% for sets A, B, and C, respectively. These results indicate that SPT is the most promising heuristic to be used in advanced metaheuristics for max-RWA. Therefore, SPT will be used as the decoder for our implementation of the BRKGA heuristic for max-RWA.

In the second experiment, we investigate if BRKGA with SPT as its decoder efficiently identifies the relationships between keys and good solutions and converges to near-optimal solutions. For this end, we compare its performance with that of a multi-start (MS) procedure that uses the same decoding heuristic as BRKGA. Each iteration of the multi-start procedure applies this same decoding heuristic starting from a randomly generated vector of random-keys. Therefore, nothing is learned from one iteration of MS to the next.

Each of the heuristics BRKGA and MS was given ten minutes of computation time and stopped thereafter. Ten runs of each heuristic have been performed for each instance, with different seeds for the random number generator [53]. The results for instance sets A, B, and C are displayed in Tables 2 to 4. The first two columns provide the network name and the upper bound (*UB*) already reported in [46]. The minimum (*min*), average (*avg*), and maximum (*avg*) solution values obtained by MS are displayed in the next three columns. The average optimality gap, and the coefficient of variation ( $CV = \sigma/avg$ ) are shown in the following two columns, where  $\sigma$  is the standard deviation of the sample. The same statistics are displayed for BRKGA in the subsequent columns. The best average optimality gaps for each instance are displayed in boldface.

Table 2 shows that BRKGA found better solutions than MS for all instances in set A. The average optimality gap observed for BRKGA was only 3.54%, while that for MS was 5.45%. Besides, the maximum gap for MS was 10.38%, while that for BRKGA was only

7.30%. Similar results were observed for sets B and C. Table 3 shows that MS never obtained better average solution values than BRKGA, which found strictly better solutions than MS for all, but three instances in set B. The average optimality gap obtained with BRKGA for this set was only 3.99%, while that resulting from MS was 5.49%. The maximum gap for MS was 9.72%, while that for BRKGA was only 8.23%. Relating to set C, BRKGA found better solutions than MS for all, but the six instances where both heuristics found the optimal solution value. The average optimality gap of BRKGA for set C was only 3.14%, while that for MS was 4.18%. The maximum gap for MS was 10.90%, while that for BRKGA was an average optimality gaps for all test sets. In addition, the coefficient of variation of BRKGA was at most 1.78% from the average (observed for network ATT2 in set C) and less than 1% for 68 out of the 72 test instances.

Run time distributions or 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 [23] as a way to characterize the running times of stochastic local search algorithms for combinatorial optimization. In the next experiment, both heuristics MS and BRKGA were made to stop whenever a solution with cost smaller than or equal to a given target value was found. This target value was set to the average cost of the solutions obtained by MS in the previous experiments, i.e., the average of ten 10-minute runs of MS for each instance. The heuristics were run 200 times on each instance, with different 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 described in [1,38]. We associate a probability  $p_i = (i - \frac{1}{2})/200$  with the *i*-th smallest running time  $t_i$  and plot the points  $z_i = (t_i, p_i)$ , for i = 1, ..., 200. The more to the left is a plot, the better is the algorithm corresponding to it.

We illustrate the runtime distributions (or ttt-plots, for short) for one instance of each test set. The runtime distributions for the instance of set A defined by network ATT2 are shown in Figure 6. BRKGA finds solutions with 856 accepted lightpaths with a probability close to 100% in less than ten seconds of running time, while MS may take up to 2,500 seconds to find solutions as good as them with the same probability. Similar results were observed for the instances of sets B and C. The ttt-plots for the instance of set B defined by network FRANCE are shown in Figure 7. BRKGA finds solutions with 1072 accepted lightpaths with a probability close to 100% in less than nine seconds of running time, while MS may take up to 5,000 seconds to find solutions as good as them with the same probability. Finally, the runtime distributions for the instance of set C defined by the network DFN-GWIN is shown in Figure 8. In this case, BRKGA finds solutions with 1944 accepted lightpaths with a probability close to 100% in less than five seconds of running time, while MS may take up to 1,500 seconds to find solutions as good as them with the same probability.

We also used the tool proposed by Ribeiro et al. [9] to perform a direct numerical comparison of heuristics MS and BRKGA. Let  $T_{BRKGA}$  (resp.  $T_{MS}$ ) be the continuous random variable representing the time needed by heuristic BRKGA (resp. MS) to find a solution as good as a given target value and let  $P(T_{BRKGA} \le T_{MS})$  be the probability that BRKGA converges faster than MS, computed by the software available in [10]. The captions of Figures 6 to 8 show that  $P(T_{BRKGA} \le T_{MS}) \ge 0.985$  for the above three instances, further illustrating the superiority of BRKGA with respect to MS.

Figures 9 and 10 illustrate the evolution of the solution population along 200 generations of BRKGA for one execution of instances DFN-GWIN and ATT2. They show that the biased

random-key genetic algorithm is able to continuously evolve the solution population and to improve the best solution value.

14

 Table 2 BRKGA vs. multi-start for instances of test set A (the smallest optimality gap for each instance is displayed in boldface).

				MS					BRKGA		
Instance	UB	min	ave	max	gap (%)	CV (%)	min	ave	max	gap (%)	CV (%)
ATT	253.00	234.00	234.60	235.00	7.27	0.11	236.00	239.80	244.00	5.22	0.31
ATT2	895.00	856.00	856.60	858.00	4.29	0.21	886.00	887.40	889.00	0.85	0.29
BRASIL	721.50	662.00	664.30	666.00	7.93	0.23	677.00	680,40	686.00	5.70	0.60
COST266	788.50	735.00	737.10	740.00	6.52	0.33	750.00	754.40	762.00	4.32	0.81
DFN-BWIN	884.00	809.00	811.10	814.00	8.25	0.35	837.00	844.10	853.00	4.51	1.08
DFN-GWIN	874.00	802.00	803.10	805.00	8.11	0.22	827.00	834.70	843.00	4.50	0.99
EON	285.00	279.00	279.30	280.00	2.00	0.11	280.00	281.20	282.00	1.33	0.13
FINLAND	444.77	408.00	408.70	411.00	8.11	0.21	415.00	420.40	424.00	5.48	0.60
FRANCE	610.00	545.00	546.70	550.00	10.38	0.31	557.00	565.50	573.00	7.30	1.28
GIUL	1,561.00	1,477.00	1,480.20	1,486.00	5.18	0.60	1,519.00	1,524.00	1,527.00	2.37	0.62
JANOS-US	600.00	584.00	585.20	587.00	2.47	0.20	587.00	591.30	594.00	1.45	0.43
NOBEL-EU	346.00	330.00	330.20	332.00	4.57	0.14	334.00	336.20	338.00	2.83	0.24
NOBEL-GERMANY	228.00	225.00	225.60	226.00	1.05	0.11	227.00	227.70	228.00	0.13	0.10
NOBEL-US	190.00	187.00	187.90	188.00	1.11	0.07	189.00	189.80	190.00	0.11	0.09
NORWAY	834.00	795.00	795.80	797.00	4.58	0.17	806.00	809.70	814.00	2.91	0.52
NSF1	197.00	186.00	186.20	187.00	5.48	0.09	187.00	187.70	189.00	4.72	0.14
NSF12	264.00	246.00	247.10	248.00	6.40	0.16	252.00	252.80	254.00	4.24	0.17
NSF21	205.00	195.00	195.60	196.00	4.59	0.11	196.00	196.70	198.00	4.05	0.14
NSF212	280.33	261.00	261.40	263.00	6.75	0.15	264.00	267.00	269.00	4.76	0.30
NSF23	206.00	194.00	194.50	195.00	5.58	0.12	196.00	197.40	199.00	4.17	0.18
NSF248	266.33	251.00	251.70	252.00	5.49	0.11	254.00	256.50	258.00	3.69	0.31
NSF3	195.50	183.00	183.10	184.00	6.34	0.07	184.00	184.90	186.00	5.42	0.01
NSF48	254.00	239.00	239.90	241.00	5.55	0.12	243.00	244.20	246.00	3.86	0.26
SUN	264.00	256.00	256.70	257.00	2.77	0.11	258.00	261.10	262.00	1.10	0.29
Average					5.45					3.54	

| 15

 Table 3 BRKGA vs. multi-start for instances of test set B (the smallest optimality gap for each instance is displayed in boldface).

				MS					BRKGA		
Instance	UB	min	avg	max	gap (%)	CV (%)	min	avg	max	gap (%)	CV (%)
ATT	359.00	324.00	324.90	326.00	9.50	0.08	331.00	333.80	336.00	7.02	0.02
ATT2	1,298.00	1,210.00	1,211.60	1,213.00	6.66	0.11	1,238.00	1,240.80	1,245.00	4.41	0.29
BRASIL	1,080.67	1,017.00	1,018.40	1,023.00	5.76	0.32	1,026.00	1,030.40	1,033.00	4.65	0.49
COST266	1,325.00	1,197.00	1,199.50	1,205.00	9.47	0.32	1,232.00	1,238.00	1,243.00	6.57	0.49
DFN-BWIN	1,733.00	1,615.00	1,618.20	1,623.00	6.62	0.33	1,657.00	1,665.70	1,671.00	3.88	0.64
DFN-GWIN	1,519.00	1,464.00	1,466.00	1,470.00	3.49	0.23	1,492.00	1,498.60	1,502.00	1.34	0.42
EON	369.00	367.00	367.00	367.00	0.54	0.00	368.00	368.40	369.00	0.16	0.07
FINLAND	642.00	607.00	607.40	608.00	5.39	0.07	609.00	613.10	617.00	4.50	0.38
FRANCE	1,181.00	1,069.00	1,072.40	1,078.00	9.20	0.34	1,097.00	1,103.20	1,111.00	6.59	0.61
GIUL	2,519.00	2,365.00	2,367.40	2,370.00	6.02	0.23	2,421.00	2,430.10	2,440.00	3.53	0.86
JANOS-US	981.00	913.00	914.70	917.00	6.76	0.16	927.00	933.90	938.00	4.80	0.45
NOBEL-EU	596.00	561.00	561.70	564.00	5.76	0.13	567.00	572.10	576.00	4.01	0.33
NOBEL-GERMANY	384.00	363.00	364.60	366.00	5.05	0.16	367.00	369.90	371.00	3.67	0.17
NOBEL-US	326.50	311.00	311.80	313.00	4.50	0.09	314.00	314.90	317.00	3.55	0.13
NORWAY	1,377.00	1,262.00	1,263.80	1,266.00	8.22	0.15	1,289.00	1,296.10	1,303.00	5.88	0.58
NSF1	278.00	272.00	272.00	272.00	2.16	0.00	272.00	272.00	272.00	2.16	0.00
NSF12	408.00	381.00	382.30	383.00	6.30	0.09	386.00	387.40	389.00	5.05	0.15
NSF21	282.00	282.00	282.00	282.00	0.00	0.00	282.00	282.00	282.00	0.00	0.00
NSF212	427.00	403.00	403.00	403.00	5.62	0.00	407.00	408.60	411.00	4.31	0.18
NSF23	284.00	280.00	280.00	280.00	1.41	0.00	280.00	280.00	280.00	1.41	0.00
NSF248	413.00	383.00	384.20	385.00	6.97	0.09	389.00	392.80	395.00	4.89	0.23
NSF3	277.00	273.00	273.00	273.00	1.44	0.00	273.00	273.30	274.00	1.34	0.06
NSF48	389.00	368.00	368.78	370.00	5.20	0.09	372.00	374.33	378.00	3.77	0.24
SUN	502.00	452.00	453.20	455.00	9.72	0.14	459.00	460.70	462.00	8.23	0.14
Average					5.49					3.99	

J. S. Brandão, T. F. Noronha, C. C. Ribeiro

 Table 4 BRKGA vs. multi-start for instances of test set C (the smallest optimality gap for each instance is displayed in boldface).

							· ·				
-				MS					BRKGA		
Instance	UB	min	avg	max	gap (%)	CV (%)	min	avg	max	gap (%)	CV (%)
ATT	359.00	359.00	359.00	359.00	0.00	0.00	359.00	359.00	359.00	0.00	0.00
ATT2	1,648.00	1,517.00	1,518.60	1,520.00	7.85	0.12	1,558.00	1,581.40	1,603.00	4.04	1.78
BRASIL	1,241.00	1,233.00	1,234.10	1,235.00	0.56	0.06	1,236.00	1,237.00	1,238.00	0.32	0.08
COST266	1,728.94	1,567.00	1,567.60	1,569.00	9.33	0.07	1,592.00	1,600.60	1,609.00	7.42	0.63
DFN-BWIN	2,555.50	2,407.00	2,409.20	2,413.00	5.72	0.17	2,442.00	2,460.20	2,478.00	3.73	1.27
DFN-GWIN	1,945.00	1,943.00	1,944.40	1,945.00	0.03	0.07	1,945.00	1,945.00	1,945.00	0.00	0.00
EON	373.00	373.00	373.00	373.00	0.00	0.00	373.00	373.00	373.00	0.00	0.00
FINLAND	774.00	748.00	749.30	752.00	3.19	0.11	752.00	754.50	760.00	2.52	0.24
FRANCE	1,730.00	1,570.00	1,573.40	1,579.00	9.05	0.29	1,598.00	1,607.00	1,615.00	7.11	0.57
GIUL	3,307.50	3,058.00	3,065.00	3,078.00	7.33	0.59	3,130.00	3,140.70	3,160.00	5.04	0.91
JANOS-US	1,239.00	1,172.00	1,172.30	1,173.00	5.38	0.05	1,187.00	1,189.50	1,193.00	4.00	0.23
NOBEL-EU	822.00	757.00	757.70	760.00	7.82	0.10	769.00	772.00	776.00	6.08	0.16
NOBEL-GERMANY	489.00	462.00	462.70	465.00	5.38	0.10	465.00	467.10	469.00	4.48	0.16
NOBEL-US	432.50	403.00	404.00	405.00	6.59	0.05	407.00	409.50	412.00	5.32	0.14
NORWAY	1,782.00	1,635.00	1,636.90	1,639.00	8.14	0.14	1,663.00	1,675.80	1,683.00	5.96	0.58
NSF1	284.00	284.00	284.00	284.00	0.00	0.00	284.00	284.00	284.00	0.00	0.00
NSF12	499.00	476.00	476.30	477.00	4.55	0.05	477.00	480.70	482.00	3.67	0.14
NSF21	284.00	284.00	284.00	284.00	0.00	0.00	284.00	284.00	284.00	0.00	0.00
NSF212	522.00	503.00	504.10	505.00	3.43	0.06	509.00	511.10	514.00	2.09	0.19
NSF23	285.00	285.00	285.00	285.00	0.00	0.00	285.00	285.00	285.00	0.00	0.00
NSF248	505.00	486.00	486.40	487.00	3.68	0.05	490.00	492.50	495.00	2.48	0.16
NSF3	285.00	285.00	285.00	285.00	0.00	0.00	285.00	285.00	285.00	0.00	0.00
NSF48	469.00	461.00	462.00	463.00	1.49	0.07	464.00	466.30	467.00	0.58	0.10
SUN	722.00	643.00	643.30	644.00	10.90	0.05	646.00	646.00	646.00	10.53	0.00
Average					4.18					3.14	

17



Fig. 6 Runtime distributions for the instance of set A defined by network ATT2 with the target value set at 856:  $P(T_{BRKGA} \le T_{MS}) = 0.985$ .



**Fig. 7** Runtime distributions for the instance of set B defined by network FRANCE with the target value set at 1072:  $P(T_{BRKGA} \le T_{MS}) = 0.995$ .

Finally, Figures 11 and 12 illustrate (for the instance of set B defined by network DFN-GWIN with 20 lightpath requests) how the best solutions found by BRKGA and by the multi-start procedure evolve along the first six and the first 60 seconds of processing time, respectively. They show that the biased random-key genetic algorithm systematically finds better solutions faster than the other algorithm. The best solution obtained by BRKGA is better than that found by multi-start at any time along the run whose results are displayed in these figures. This same behavior illustrated for this run is observed for all test instances.

These results show that BRKGA identifies the best relationships between the keys and the good solutions throughout the evolutionary process, converging to high-quality, nearoptimal solutions.



**Fig. 8** Runtime distributions for the instance of set C defined by network DFN-GWIN with the target value set at 1944:  $P(T_{BRKGA} \le T_{MS}) = 0.996$ .



**Fig. 9** Population evolution for the instance of set B defined by network DFN-GWIN: the value of the best solution found by BRKGA after 4.42 seconds (200 generations) is 1492, while the best solution value after 10 minutes of running time is 1502.

In the third experiment, we compare the quality of the solutions provided by BRKGA and the tabu search (TS) heuristic for the instance set D, which was used by Dzongang et al. [13] to evaluate the performance of their heuristic. Upper bounds for the 30 instances are known from [34]. The tabu search heuristic was implemented in C++ and the experiments were performed in a Pentium 4 with 2.4 GHz. Individual running times for each instances are not reported. However, it is said that the tabu search run for up to 60 seconds. Our implementation of BRKGA was run 10 times for each instance, with different seeds for the random number generator [53]. The stopping criterion of BRKGA was set to 60 seconds, in order to make its running times compatible with those of the tabu search heuristic.

The results for this experiment are displayed in Table 5. The first column provides the number ( $\lambda$ ) of wavelengths available for the instance. The next three columns display the upper bound, the optimality gap of TS, and the average optimality gap of BRKGA for the corresponding NSF instance with  $\lambda$  wavelengths. The last three columns show the upper



Fig. 10 Population evolution for the instance of set A defined by network ATT2 the value of the best solution found by BRKGA after 42.00 seconds (200 generations) is 883, while the best solution value after 10 minutes of running time is 889.



Fig. 11 Evolution of the best solutions found by BRKGA and MS along the six first seconds of processing time for the instance of set B defined by network DFN-GWIN with 20 lightpaths: the best solution value obtained by BRKGA is 1493, while that found by MS is only 1457.

bound, the optimality gap of TS, and the average optimality gap of BRKGA for the corresponding EONNET instance with  $\lambda$  wavelengths. The smallest optimality gap for each instance is displayed in boldface. Regarding the 15 instances with 14 nodes and 21 links based on NSF, TS found smaller optimality gaps on 8 instances, while BRKGA out performs TS on five instances. The average optimality gap of TS and BRKGA was 1.41% and 2.50%, respectively. However, BRKGA found optimal solutions for six (out of the 15) instances, while TS found optimal solutions for only two instances. Regarding the 15 largest instances with 20 nodes and 39 links based on EONNET, BRKGA found solutions better than those of TS for 13 instances, while both heuristics found the optimal solution for the other two instances. The average optimality gap of TS and BRKGA was 1.53% and 0.62%, respectively. For the 30 instances in this set together, The average optimality gap of TS



Fig. 12 Evolution of the best solution values found by BRKGA and MS along the 60 first seconds of processing time for the instance of set B defined by network DFN-GWIN with 20 lightpaths: the best solution value obtained by BRKGA is 1499, while that found by MS is only 1461.

and BRKGA was 1.47% and 1.56%, respectively. These results show that BRKGA finds solutions competitive with those of the tabu search heuristic of [13].

Table 5 BRKGA vs. the Tabu Search (TS) of [13] for instances of test set D. The smallest optimality gap for each instance is displayed in boldface.

		NS	F		EON	NET
λ	UB	TS (%)	BRKGA (%)	UB	TS (%)	BRKGA (%)
10	198	0.51	4.65	285	1.40	1.23
11	218	4.59	8.49	301	2.33	1.43
12	218	0.00	3.62	317	3.15	1.58
13	238	4.20	7.65	329	3.34	1.40
14	238	0.42	2.98	337	2.67	0.86
15	248	1.21	3.27	344	1.74	0.58
16	258	2.33	3.53	350	1.43	0.49
17	263	1.90	2.05	356	1.12	0.53
18	267	2.25	1.24	362	1.66	0.47
19	268	1.49	0.00	367	1.63	0.19
20	268	1.12	0.00	370	1.08	0.22
21	268	0.75	0.00	373	0.80	0.29
22	268	0.37	0.00	374	0.53	0.03
23	268	0.00	0.00	374	0.00	0.00
24	268	0.00	0.00	374	0.00	0.00
Average:		1.41	2.50		1.53	0.62

#### 6 Conclusions

We have shown that most of the work in the literature about the max-RWA version of the problem of routing and wavelength assignment is based on integer programming formulations and focus into finding exact solutions or upper bounds to the optimal value. Only small instances with no more than 27 nodes are solved to optimality, and only upper bounds are known for larger instances with up to 90 nodes in sets A, B, and C.

We proposed new greedy constructive heuristics and a biased random-key genetic algorithm. Computational experiments showed that the greedy heuristic SPT outperformed the best greedy heuristics in literature. In addition, the biased random-key genetic algorithm using SPT as the decoding heuristic found near-optimal solutions with average optimality gaps of 3.54%, 3.99%, 3.14%, and 1.56% for the instances in set A, B, C, and D, respectively. The optimality gap of this heuristic was at most 10.53% over all 102 test instances, observed for network SUN in test set C.

## References

- Aiex, R., Resende, M., Ribeiro, C.C.: Probability distribution of solution time in GRASP: An experimental investigation. Journal of Heuristics 8, 343–373 (2002)
- 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, 903–908 (1996)
- Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 2, 154–160 (1994)
- Belgacem, L., Puech, N.: Solving large size instances of the RWA problem using graph partitioning. In: International Conference on Optical Network Design and Modeling. IEEE, Barcelona (2008)
- 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 Operational Research 22, 823–839 (2015)
- Buriol, L.S., Hirsch, M.J., Pardalos, P.M., Querido, T., Resende, M.G.C., Ritt, M.: A biased random-key genetic algorithm for road congestion minimization. Optimization Letters 4, 619–633 (2010)
- Buriol, L.S., Resende, M.G.C., Ribeiro, C.C., Thorup, M.: A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. Networks 46, 36–56 (2005)
- Buriol, L.S., Resende, M.G.C., Thorup, M.: Survivable IP network design with OSPF routing. Networks 49, 51–64 (2007)
- C., R.C., I., R., R., V.: Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. Journal of Global Optimization 54, 405–429 (2012)
- C., R.C., I., R., R., V.: tttplots-compare: A perl program to compare time-to-target plots or general runtime distributions of randomized algorithms (2014). Online reference at http://www2.ic.uff.br/~celso/tttplots/index.html, last visited on January 25, 2014.
- Chen, C., Banerjee, S.: A new model for optimal routing and wavelength assignment in wavelength division multiplexed optical networks. In: Proceedings of the Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation, vol. 1, pp. 64–171. San Francisco (1996)
- Choi, J.S., Golmie, N., Lapeyrere, F., Mouveaux, F., Su, D.: A functional classification of routing and wavelength assignment schemes in DWDM networks: Static case. In: Proceedings of the 7th International Conference on Optical Communication and Networks, pp. 1109–1115. Paris (2000)
- Dzongang, C., Galinier, P., Pierre, S.: A tabu search heuristic for the routing and wavelength assignment problem in optical networks. IEEE Communications Letters 9, 426–428 (2005)
- 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, 299–333 (2002)
- Erlebach, T., Jansen, K.: The complexity of path coloring and call scheduling. Theoretical Computer Science 255, 33–50 (2001)
- F., N.T., Ribeiro, C.C.: Routing and wavelength assignment by partition coloring. European Journal of Operational Research 171, 797–810 (2006)
- Gonçalves, J.F., de Magalhães Mendes, J.J., Resende, M.G.C.: A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research 167, 77–95 (2005)
- Gonçalves, J.F., Resende, M.G.C.: Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics 17, 487–525 (2011)
- Gonçalves, J.F., Mendes, J.J.M.: A hybrid genetic algorithm for assembly line balancing. Journal of Heuristics 8, 629–642 (2002)
- Gonçalves, J.F., Resende, M.G.C.: An evolutionary algorithm for manufacturing cell formation. Computers and Industrial Engineering 47, 247–273 (2004)

- Goulart, N., de Souza, S.R., Dias, L.G.S., Noronha, T.F.: Biased random-key genetic algorithm for fiber installation in optical network optimization. In: Proceedings of the 2011 IEEE Congress on Evolutionary Computation, pp. 2267–2271. New Orleans (2011)
- Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics 17, 416–429 (1969)
- Hoos, H., Stützle, T.: Evaluation of Las Vegas algorithms Pitfalls and remedies. In: G. Cooper, S. Moral (eds.) Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, pp. 238–245. Madison (1998)
- Hou, E.S., Ansari, N., Ren, H.: A genetic algorithm for multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 5(2), 113–120 (1994)
- Hyytiä, E., Virtamo, J.: Wavelength assignment and routing in WDM networks. In: Fourteenth Nordic Teletraffic Seminar, pp. 31–40. Copenhagen (1998)
- Hyytiä, E., Virtamo, J.: Wavelength assignment in multifibre in WDM-networks. Tech. Rep. COST257TD(99)04, Helsinki University of Technology (1999)
- Jaumard, B., Meyer, C., Thiongane, B.: Ilp formulations for the routing and wavelength assignment problem: Symmetric systems. In: Handbook of optimization in telecommunications, pp. 637–677. Springer (2006)
- Jaumard, B., Meyer, C., Thiongane, B.: Comparison of ILP formulations for the RWA problem. Optical Switching and Networking 4, 157–172 (2007)
- Jaumard, B., Meyer, C., Thiongane, B.: On column generation formulations for the RWA problem. Discrete Applied Mathematics 157, 1291–1308 (2009)
- Jaumard, B., Meyer, C., Thiongane, B., Yu, X.: ILP formulations and optimal solutions for the RWA problem. In: Proceedings of the IEEE Global Telecommunications Conference, vol. 3, pp. 1918–1924. Dallas (2004)
- Jaumard, B., Meyer, C., Yu, X.: How much wavelength conversion allows a reduction in the blocking rate? Journal of Optical Networking 5, 81–900 (2006)
- Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R., Graham, R.L.: Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM Journal on Computing 3, 299–325 (1974)
- 33. Kleinberg, J.: Approximation algorithms for disjoint paths problems. Ph.D. thesis, MIT, Cambridge (1996)
- Krishnaswamy, R., Sivarajan, K.: Algorithms for routing and wavelength assignment based on solutions of LP-relaxation. IEEE Communications Letters 5, 435–437 (2001)
- Lee, T., Lee, K., Park, S.: Optimal routing and wavelength assignment in wdm ring networks. IEEE Journal on Selected Areas in Communications 18, 2146–2154 (2000)
- Lei, D.: Fuzzy job shop scheduling problem with availability constraints. Computers & Industrial Engineering 58, 610–617 (2010)
- 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, pp. 1–19. Dallas (2000)
- M., A.R., Resende, M.G.C., C., R.C.: TTTPLOTS: A Perl program to create time-to-target plots. Optimization Letters 1, 355–366 (2007)
- M. Grtschel L. Lovsz, A.S.: Geometric Algorithms and Combinatorial Optimization, vol. 2. Algorithms and Combinatorics, Springer-Verlag (1993)
- Malve, S., Uzsoy, R.: A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families. Computers & Operations Research 34, 3016–3028 (2007)
- Manohar, P., Manjunath, D., Shevgaonkar, R.K.: Routing and wavelength assignment in optical networks from edge disjoint path algorithms. IEEE Communications Letters 5, 211–213 (2002)
- Marković, G., Aćimović-Raspopović, V.: Solving the RWA problem in WDM optical networks using the BCO meta-heuristic. Telfor Journal 2, 43–48 (2010)
- 43. Martins, A.X.: Metaheurísticas e formulações para a resolução do problema de roteamento e alocação de comprimentos de onda em redes ópticas. Ph.D. thesis, Universidade Federal de Minas Gerais (2011)
- Martins, A.X., Duhamel, C., De Souza, M.C., Saldanha, R.R., Mahey, P.: A VND-ILS heuristic to solve the RWA problem. In: J. Pahl, T. Reiners, S. Vo (eds.) Network Optimization, pp. 577–582. Springer (2011)
- Martins, A.X., Duhamel, C., Mahey, P., Saldanha, R.R., de Souza, M.C.: Variable neighborhood descent with iterated local search for routing and wavelength assignment. Computers & Operations Research 39, 2133–2141 (2012)
- Martins, A.X., Duhamel, C., Mahey, P., de Souza, M.C., Saldanha, R.R.: Geração de colunas para o problema de roteamento e atribuição de comprimentos de onda. In: Anais do XLIV Simp. Bras. Pesq. Operacional, pp. 1–12. Rio de Janeiro (2012)

- Noronha, T.F., Resende, M.G.C., Ribeiro, C.C.: Efficient implementation of heuristics for routing and wavelength assignment. Lecture Notes in Computer Science 5038, 169–180 (2008)
- Noronha, T.F., Resende, M.G.C., Ribeiro, C.C.: A biased random-key genetic algorithm for routing and wavelength assignment. Journal of Global Optimization 50, 503–518 (2011)
- Qin, H., Liu, Z., Zhang, S., Wen, A.: Routing and wavelength assignment based on genetic algorithm. Communications Letters 6(10), 455–457 (2002)
- Ramaswami, R., Sivarajan, K.: Optimal routing and wavelength assignment in all-optical networks. In: Proc. of IEEE INFOCOM Conference on Computer Communications, pp. 970–979. Toronto (1994)
   Reis, R., Ritt, M., Buriol, L.S., Resende, M.G.C.: A biased random-key genetic algorithm for OSPF and DEPENDENCE.
- Reis, R., Ritt, M., Buriol, L.S., Resende, M.G.C.: A biased random-key genetic algorithm for OSPF and DEFT routing to minimize network congestion. International Transactions in Operational Research 18, 401–423 (2011)
- Roque, L., Fontes, D., Fontes, F.: A hybrid biased random key genetic algorithm approach for the unit commitment problem. Journal of Combinatorial Optimization 28(1), 140–166 (2014)
- Schrage, L.: A more portable Fortran random number generator. ACM Transactions on Mathematical Software 5, 132–138 (1979)
- 54. Skorin-Kapov, N.: Routing and wavelength assignment in optical networks using bin packing based algorithms. European Journal of Operational Research **177**, 1167–1179 (2007)
- Spears, W., deJong, K.: 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. Morgan Kaufman, San Mateo (1991)
- Wang, C.S., Uzsoy, R.: A genetic algorithm to minimize maximum lateness on a batch processing machine. Computers & Operations Research 29, 1621–1640 (2002)
- 57. Zang, H., Jue, J.P., Mukherjee, B.: A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks. Optical Networks Magazine 1, 47–60 (2000)
- Zheng, J.N., Chien, C.F., Gen, M.: Multi-objective multi-population biased random-key genetic algorithm for the 3-d container loading problem. Computers & Industrial Engineering (2014)

ANEXO B - Brandão, J.S., Noronha, T.F., Resende, M.G.C., Ribeiro, C.C. "A biased random-key genetic algorithm for scheduling divisible loads". In Proceedings of the Multidisciplinary International Scheduling Conference: Theory and Applications (Prague, 2015), p. 570 - 578. MISTA 2015

# A biased random-key genetic algorithm for scheduling divisible loads

Julliany S. Brandão · Thiago F. Noronha · Mauricio G. C. Resende · Celso C. Ribeiro

Abstract A divisible load is an amount  $W \ge 0$  of computational work that can be arbitrarily divided into chunks and distributed among a set P of worker processors to be processed in parallel. The Divisible Load Scheduling Problem consists in (a) selecting a subset of active workers, (b) defining the order in which the chunks will be transmitted to each of them, and (c) deciding the amount of load  $\alpha_i$  that will be transmitted to each active worker, so as to minimize the makespan, i.e., the total elapsed time since the master began to send data to the first worker, until the last worker stops its computations. We propose a biased random-key genetic algorithm for solving the divisible load scheduling problem. Computational results show that the genetic algorithm outperforms the best heuristic in the literature.

**Keywords** Divisible load scheduling, Random-key genetic algorithms, parallel processing, scientific computing

Julliany S. Brandão

Thiago F. Noronha

Mauricio G. C. Resende

Mathematical Optimization and Planning, Amazon.com 333 Boren Avenue North, Seattle, WA 98109, USA (work of this author was done when he was employed by AT&T Labs Research) E-mail: resendem@amazon.com

Celso C. Ribeiro

Universidade Federal Fluminense, Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil E-mail: celso@ic.uff.br

Universidade Federal Fluminense, Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil, and Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, Av. Maracanã 229, Rio de Janeiro, RJ 20271-110, Brazil E-mail: jbrandao@ic.uff.br

Universidade Federal de Minas Gerais, Avenida Antônio Carlos 6627, Belo Horizonte, MG 24105, Brazil E-mail: tfn@dcc.ufmg.br

#### 1 Introduction

A divisible load is an amount  $W \ge 0$  of computational work that can be arbitrarily divided and distributed among different processors to be processed in parallel. The processors are arranged in a star topology and the load is stored in a central *master* processor. The latter splits the load into chunks of arbitrary sizes and transmits each of them to other processors, called *workers*. The master does not process the load itself.

The master can only send load to one worker at a time. Any worker can only start processing after it has completely received its respective chunk of the load. The workers are heterogeneous in terms of processing power, communication speed, and setup time for start communicating with the master. Not all available processors should necessarily be used for processing the load.

The Divisible Loading Scheduling Problem (DLSP) was introduced in [12], motivated by an application in intelligent sensor networks. Applications of DLSP arise from a number of scientific problems, such as parallel database searching [11], parallel image processing [25], parallel video encoding [26,35], processing of large distributed files [37], and task scheduling in cloud computing [28], among others.

In this work, we deal with the same DLSP variant treated in [1, 10]. Let  $P = \{1, ..., n\}$  be the set of worker processors indices. Each worker  $i \in P$  has (i) a setup time  $g_i \geq 0$  to start the communication with the master, (ii) a communication time  $G_i \geq 0$  needed to receive each unit of the load from the master and (iii) a processing time  $w_i \geq 0$  needed to process each unit of the load. Therefore, it takes  $g_i + \alpha_i \cdot G_i$  units of time for the master to transmit a load chunk of size  $\alpha_i \geq 0$  to the worker  $i \in P$ . Furthermore, it takes an additional  $w_i \cdot \alpha_i$  units of time for this worker to process the chunk of load assigned to it.

The scheduling problem consists of (a) selecting a subset  $A \subseteq P$  of active workers, (b) defining the order in which the chunks will be transmitted to each active worker and (c) deciding the amount of load  $\alpha_i$  that will be transmitted to each worker  $i \in A$ , so as to minimize the makespan, i.e., the total elapsed time since the master began to send data to the first worker, until the last worker stops its computations. This problem was proved to be NP-hard in [42].

Since the load can be split arbitrarily, all workers stop at the same time in the optimal solution [5]. In addition, if the order in which the chunks are transmitted to the workers is fixed, then the best solution can be computed in O(n) time, using the AlgRap algorithm developed in [1]. In other words, given a permutation of the workers in P, AlgRap computes the set of active workers and the amount of load that has to be sent to each of them to minimize the makespan. Therefore, DLSP can be reduced to the problem of finding the best permutation of the processors, i.e., the one which induces the minimum makespan. In order to find this permutation, we propose a biased random-key genetic algorithm [18, 19, 22], which has been successfully used for solving many permutation based combinatorial optimization problems [17, 19–21, 23, 30].

The paper is organized as follows. Related work is reviewed in the next section. The proposed heuristic is described in Section 3. Computational experiments are reported and discussed in Section 4. Concluding remarks are drawn in the last section.

#### 2 Related work

There are many variants of DLSP in the literature. Divisible load scheduling may be performed in a *single round* or in *multiple rounds*. In the single round case [1, 3-5, 7, 9, 10, 13, 15, 16, 24, 27, 32, 36, 38, 39], each active worker receives and processes a single chunk of the load. In the multi-round case [1, 4, 6, 8, 14-16, 31, 33, 39-41], each active worker receives and processes multiple chunks of load. After the master finishes the transmission of the first round of load to all active workers, it immediately starts the transmission of the next batch of load chunks following the same order, until all the load is distributed among the workers.

The workers may be homogeneous or heterogeneous. If the workers are homogeneous, the values of  $g_i$ ,  $G_i$ , and  $w_i$  are the same for all processors  $i \in P$  [4,8,9,24,40,41]. Contrarily, in the heterogeneous case the values of  $g_i$ ,  $G_i$ , and  $w_i$  may be different for each worker [1,3,5,6,10,13–16,27,31–33,36,38–41].

The system can be *dedicated* or *non-dedicated*. When the system is dedicated, it is assumed that all resources (processors, memory, network, etc.) are used to process a single computational load [1,5-7,9,10,13,15,16,31,36,38-41]. Non-dedicated systems may be used to simultaneously process different computational loads [6,7,32].

In this work, we consider the single round DLSP with dedicated and heterogeneous workers [1, 4, 5, 10, 15, 38, 42], that was proved to be *NP*-hard in [42]. Blazewicz and Drozdowski [10] showed that once a permutation of the workers is given, a solution with minimum makespan can be obtained in  $O(n \log n)$  time, where n = |P| is the number of workers. Later, Abib and Ribeiro [1] proposed the faster AlgRap algorithm that finds this solution in O(n) time.

Non-linear programming formulations for the single round DLSP with dedicated and heterogeneous workers was proposed in [13]. The first mixed integer linear programming formulation was proposed in [4] and was later improved and extended in [1]. Computational experiments reported in [1] have shown that the CPLEX branch-andcut algorithm based on this formulation was able to find optimal solutions for 490 out of 720 instances with up to 160 processors. However, for the largest unsolved instances, the computational gaps were very high, which motivated the development of heuristics for solving DLSP.

To the best of our knowledge, the best heuristic in the literature for the DLSP variant studied in this paper is the HeuRet algorithm of Abib and Ribeiro [1]. At each iteration, their algorithm (i) estimates the *performance*  $e_i$  of each worker in  $i \in P$  and (ii) builds a solution by taking the processors in P in a non-increasing order of the  $e_i$  values. The algorithm sets  $e_i = G_i$ , for all  $i \in P$ , in the first iteration and makes use of the exact AlgRap algorithm to compute an initial solution  $s_0$  with makespan  $T_0$ . Next, at each forthcoming iteration k, the estimated performance  $e_i$  of each worker  $i \in P$  is updated from the values of  $g_i$ ,  $w_i$ ,  $G_i$ , and  $T_{k-1}$  and a new solution with makespan  $T_k$  is built by algorithm AlgRap considering the new order defined on the workers. The procedure stops when  $T_{k-1} < T_k$ , i.e. when the new solution degenerates the makespan of the previous one.

The heuristic proposed in the next section improves upon HeuRet because, instead of defining a greedy local search on the performance estimation of the workers, it performs a global search in the space of worker permutations in order to find the permutation that induces the optimal or a near-optimal solution.

# 3 Biased random-key genetic algorithm

Genetic algorithms with random keys, or random-key genetic algorithms (RKGA), were first introduced in [2] for combinatorial optimization problems whose solutions can be represented by a permutation vector. 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 a RKGA. Parents are allowed to be selected for mating more than once in a given generation.

A biased random-key genetic algorithm (BRKGA) differs from a RKGA in the way parents are selected for crossover [19]. 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 biased because one parent is always an elite individual and has a higher probability of passing its genes to the new generation.

The BRKGA-DLS biased random-key genetic algorithm for DLSP evolves a population of chromosomes that consists of vectors of real numbers (keys). Each solution is represented by a |P|-vector, in which each component is a real number in the range [0, 1] associated with a worker processor in P. Each solution represented by a chromosome is decoded by a heuristic that receives the vector of keys and builds a feasible solution for DLSP using algorithm AlgRap [1]. Decoding consists of two steps: first, the processors are sorted in a non-decreasing order of their random keys; next, the resulting order is used as the input for AlgRap. The makespan of the solution provided by AlgRap is used as the fitness of the chromosome.

We use the parametric uniform crossover scheme proposed in [34] to combine two parent solutions and produce an offspring. In this scheme, the offspring inherits each of its keys from the best fit of the two parents with probability 0.60 and from the least fit parent with probability 0.40. Instead of the mutation operator, the concept of mutants is used: a fixed number of mutant solutions are introduced in the population in each generation, randomly generated in the same way as in the initial population.

The keys associated to each worker are randomly generated in the initial population. At each generation, the population is partitioned into two sets: TOP and REST. Consequently, 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 on 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. 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 [2], 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. |MID| = |REST| - |BOT| offspring solutions are created. In our implementation, the population size was set to  $|TOP| + |MID| + |BOT| = 5 \times |P|$ , with the sizes of sets TOP, MID, and BOT set to  $0.15 \times 5 \times |P|$ ,  $0.7 \times 5 \times |P|$ , and  $0.15 \times 5 \times |P|$ , respectively, as suggested by Noronha et al. [30].



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

# 4 Computational experiments

We report computational experiments to assess the performance of the biased randomkey genetic algorithm BRKGA-DLS. This algorithm was implemented in C++. The experiments were performed on a Quad-Core AMD Opteron(tm) Processor 2350, with 16 GB of RAM memory.

The set of instances used in the experiments was proposed in [1]. There are 120 grid configurations with n = 10, 20, 40, 80, 160 worker processors and eight combinations of the parameter values  $g_i$ ,  $G_i$  and  $w_i$ ,  $i = 1, \ldots, n$ , each of them ranging either in the interval [1, 100] or in the interval [1000, 100000]. Three instances were generated for each combination of  $n, g_i, G_i$ , and  $w_i$ . Six different values of the load W = 100, 200, 400, 800, 1600, 3200 are considered for each grid configuration, corresponding to 18 instances for each combination of parameters  $g_i, G_i$  and  $w_i$ , and amounting to a total of 720 test instances. Each heuristic was run ten times for each instance, with different seeds for the random number generator of [29].

The first experiment consisted in evaluating if BRKGA-DLS efficiently identifies the relationships between keys and good solutions and converges faster to near-optimal solutions. We compare its performance with that of a multi-start procedure that uses the same decoding heuristic as BRKGA-DLS. Each iteration of the multi-start procedure, called MS-DLS, applies the same decoding heuristic of BRKGA-DLS, but using randomly generated values for the keys. In this experiment, BRKGA-DLS was run for 1000 generations and MS-DLS for  $1000 \times q$  iterations, where  $q = 5 \times |P|$  is the population size of BRKGA-DLS. The average solution values found by BRKGA-DLS were 4.95% better than those provided by MS-DLS over the 720 instances. Also, the worst (resp. best) solution values found by BRKGA-DLS were 5.98% (resp. 3.78) smaller than the respective worst (resp. best) solution values obtained with MS-DLS. These results indicate that BRKGA-DLS identifies the relationships between keys and good solutions, making the evolutionary process converge to better solutions faster than MS-DLS.

In the second experiment, we compared BRKGA-DLS with HeuRet and MS-DLS. We first evaluated how many optimal solutions have been obtained by each heuristic
over the 720 test instances. The default CPLEX branch-and-cut algorithm based on the formulation in [1] was also run for the 720 instances. Version 12.6 of CPLEX was used and the maximum CPU time was set to 24 hours. CPLEX was able to prove the optimality for 497 out of the 720 test instances. The first line of Table 1 shows that BRKGA-DLS found optimal solutions for 413 instances (i.e. 83.1%) out of the 497 instances for which the optimal solutions are known, while HeuRet found 320 optimal solutions and MS-DLS found only 177 of them. The second line of Table 1 gives the number of instances for which each heuristic found the best known solution value. The third line of this table shows the number of runs for which BRKGA-DLS and MS-DLS found the best known solution values (we recall that HeuRet is a deterministic algorithm, while the others are randomized). Finally, the last line of this table gives, for each of the three heuristics, a score that represents the sum over all instances of the number of methods that found strictly better solutions than the specific heuristic being considered. The lower a score is, the best the corresponding heuristic is. It can be seen that BRKGA outperformed both HeuRet and MS-DLS heuristics with respect to the number of optimal and best solutions found, as well as with respect to the score value. In particular, BRKGA-DLS obtained better scores than HeuRet for all but one instance and found the best known solution values for 645 out of the 720 test instances, while HeuRet found the best solution values for only 313 instances.

 ${\bf Table 1} \ {\rm Summary \ of \ the \ numerical \ results \ obtained \ with \ BRKGA-DLS, \ HeuRet \ and \ MS-DLS \ for \ 720 \ test \ instances.$ 

	MS-DLS	HeuRet	BRKGA-DLS
Optimal values (over 497 instances)	177	320	413
Best values (over 720 instances)	189	313	645
Best values (over 7200 runs)	2166	-	6191
Score value	803	112	1

The third and last experiment provides a more detailed comparison between HeuRet and BRKGA-DLS, based on 20 larger and more realistic instances with |P| = 320 and W = 10,000. The values of  $G_i$  and  $g_i$  have been randomly generated in the ranges [1, 100] and [100, 100, 000], respectively. However, differently from [1], the values of  $w_i$ have been randomly generated in the interval [200, 500]. These values are more realistic, since the processing rate of a real computer is always larger than its communication rate. In this experiment, BRKGA-DLS was made to stop after |P| generations without improvement in the best solution found. The results are reported in Table 2. The first column shows the instance name. The second and third columns display, respectively, the makespan and the computation time (in seconds) obtained by HeuRet. The next two columns provide the average makespan over ten runs of BRKGA, the corresponding coefficient of variation, defined as the ratio of the standard deviation to the average. The average computation time in seconds of BRKGA over ten runs is given in the sixth column. The last column shows the percent relative reduction between the average solution found by BRKGA-DLS with respect to that found by MS-DLS. It can be seen that the average makespan obtained by BRKGA-DLS is always smaller than that given by HeuRet. In addition, the coefficient of variation of BRKGA-DLS is very small, indicating that it is a robust heuristic. The percent relative reduction of BRKGA-DLS with respect to MS-DLS amounted to 3.19% for instance dls.320.10 and to 2.38% on average.

As in real-life applications the load size may be very large, and the total communication and processing times may take many hours, reductions in the elapsed time of this magnitude may be very significant. Although the running times of BRKGA-DLS are larger than those of HeuRet, their average values never exceeded the time taken by HeuRet by more than 30 seconds. Since practical applications of parallel processing take long elapsed times, the trade-off between the reduction in the elapsed time and this small additional running time needed by BRKGA accounts very favorably to BRKGA-DLS. In addition, we notice that the BRKGA-DLS genetic algorithm itself may be efficiently parallelized with a high speed-up and distributed over the set P of processors, making its computation time irrelevant when compared with that of the parallel application.

Table 2 BRKGA vs. HeuRet on the largest instances with 320 processors.

	Her	uRet		BRKGA		
Instance	makespan	time (s)	makespan	CV (%)	time (s)	reduction (%)
dls.320.01	312813.64	0.01	306613.22	0.04	27.24	1.98
dls.320.02	321764.07	0.01	313847.15	0.07	16.72	2.46
dls.320.03	402264.85	0.01	392059.46	0.11	23.72	2.54
dls.320.04	348474.15	0.01	341436.89	0.02	28.16	2.02
dls.320.05	342086.46	0.01	334946.37	0.03	21.67	2.09
dls.320.06	311824.17	0.01	305601.28	0.02	21.93	2.00
dls.320.07	325732.30	0.01	316467.42	0.02	23.19	2.84
dls.320.08	323171.95	0.01	315065.11	0.03	26.23	2.51
dls.320.09	312326.77	0.01	305948.81	0.02	25.03	2.04
dls.320.10	296984.47	0.01	287521.34	0.12	24.04	3.19
dls.320.11	290559.21	0.01	284822.15	0.04	20.56	1.97
dls.320.12	343076.56	0.01	333085.72	0.05	19.53	2.91
dls.320.13	287276.21	0.01	281525.27	0.04	22.77	2.00
dls.320.14	311054.47	0.01	303796.42	0.06	26.81	2.33
dls.320.15	362369.67	0.01	352642.18	0.04	20.41	2.68
dls.320.16	287083.60	0.01	281082.29	0.09	25.38	2.09
dls.320.17	339666.43	0.01	329893.61	0.04	23.53	2.88
dls.320.18	368795.14	0.01	361281.06	0.07	22.08	2.04
dls.320.19	347671.73	0.01	338075.70	0.03	27.59	2.76
dls.320.20	372427.24	0.01	364013.40	0.06	18.28	2.26
Averages	330371.15	0.01	322486.24	0.05	23.24	2.38

#### 5 Conclusions

We considered the single round divisible load scheduling problem with dedicated and heterogeneous workers. A new biased random-key genetic algorithm has been proposed for the problem.

The BRKGA-DLS heuristic improves upon the best heuristic in the literature, since it performs a global search in the space of worker permutations in order to find the best order in which the active worker processors will receive load from the master.

Computational experiments on 720 test instances with up to 160 processors have shown that BRKGA-DLS found optimal solutions for 413 instances (out of the 497 instances where the optimal solution is known), while the HeuRet heuristic found optimal solutions for only 320 of them. Moreover, BRKGA-DLS obtained better scores than HeuRet for all but one instance and found solutions as good as the best known solution for 645 out of the 720 test instances. To summarize, BRKGA-DLS outperformed the previously existing HeuRet heuristic with respect to all measures considered in Table 1.

For the new set of larger and more realistic instances with 320 processors, BRKGA-DLS found solution values 2.38% better than HeuRet on average. In addition, the processing times of BRKGA-DLS are relatively small and never exceeded 30 seconds. Therefore, parallel processing applications dealing with large amounts of data and taking long elapsed times can benefit from BRKGA-DLS, since the additional running time needed by BRKGA-DLS may result in a significant reduction in the makespan.

#### References

- 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, pp. 54–61. Nashville (2009)
- Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 2, 154–160 (1994)
- Beaumont, O., Bonichon, N., Eyraud-Dubois, L.: Scheduling divisible workloads on heterogeneous platforms under bounded multi-port model. In: International Symposium on Parallel and Distributed Processing, pp. 1–7. IEEE, Miami (2008)
- Beaumont, O., Casanova, H., Legrand, A., Robert, Y., Yang, Y.: Scheduling divisible loads on star and tree networks: results and open problems. IEEE Transactions on Parallel and Distributed Systems 16, 207–218 (2005)
- Beaumont, O., Legrand, A., Robert, Y.: Optimal algorithms for scheduling divisible workloads on heterogeneous systems. In: 12th Heterogeneous Computing Workshop, pp. 98–111. IEEE Computer Society Press, Nice (2003)
- Berlinska, J., Drozdowski, M.: Heuristics for multi-round divisible loads scheduling with limited memory. Parallel Computing 36, 199–211 (2010)
- Berlińska, J., Drozdowski, M., Lawenda, M.: Experimental study of scheduling with memory constraints using hybrid methods. Journal of Computational and Applied Mathematics 232, 638–654 (2009)
- Bharadwaj, V., Ghose, D., Mani, V.: Multi-installment load distribution in tree networks with delays. IEEE Transactions on Aerospace and Electronic Systems **31**, 555–567 (1995)
   Bharadwai, V., Chara, D., Mari, V., Pakartarai, T.C., Sakakhira distribution do in new second second
- Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.G.: Scheduling divisible loads in parallel and distributed systems. IEEE Computer Society Press (1996)
   Distributed systems of divisible integration of divisible with computer time.
- Blazewicz, J., Drozdowski, M.: Distributed processing of divisible jobs with communication startup costs. Discrete Applied Mathematics 76, 21–41 (1997)
- Błażewicz, J., Drozdowski, M., Markiewicz, M.: Divisible task scheduling-concept and verification. Parallel Computing 25, 87–98 (1999)
- Cheng, Y.C., Robertazzi, T.G.: Distributed computation with communication delay. IEEE Transactions on Aerospace and Electronic Systems 24, 700–712 (1988)
- Drozdowski, M.: Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems. 321. Politechnika Poznanska (1997)
- Drozdowski, M., Lawenda, M.: Multi-installment divisible load processing in heterogeneous systems with limited memory. Parallel Processing and Applied Mathematics 3911, 847– 854 (2006)
- Drozdowski, M., Wolniewicz, P.: Divisible load scheduling in systems with limited memory. Cluster Computing 6, 19–29 (2003)
- Drozdowski, M., Wolniewicz, P.: Optimum divisible load scheduling on heterogeneous stars with limited memory. European Journal of Operational Research 172, 545–559 (2006)
- Duarte, A., Mart, R., Resende, M., Silva, R.: Improved heuristics for the regenerator location problem. International Transactions in Operational Research 21, 541–558 (2014)
- 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, 299–333 (2002)
- Gonçalves, J.F., Resende, M.G.: Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics 17, 487–525 (2011)

ANEXO C - 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 Operational Research 22 (2015), p. 823 - 839.

# A biased random-key genetic algorithm for single-round divisible load scheduling

Julliany S. Brandão<sup>1</sup>,<sup>2</sup> \*, Thiago F. Noronha<sup>3\*</sup>, Mauricio G. C. Resende<sup>4\*\*</sup>, and Celso C. Ribeiro<sup>1\*</sup>

 <sup>1</sup> Universidade Federal Fluminense Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil {jbrandao,celso}@ic.uff.br
 <sup>2</sup> Centro Federal de Educação Tecnológica Celso Suckow da Fonseca Av. Maracanã, 229, Rio de Janeiro, RJ 20271-110, Brazil <sup>3</sup> Universidade Federal de Minas Gerais Avenida Antônio Carlos 6627, Belo Horizonte, MG 24105, Brazil tfn@dcc.ufmg.br
 <sup>4</sup> Mathematical Optimization and Planning, Amazon.com 333 Boren Avenue North, Seattle, WA 98109, USA resendem@amazon.com

**Abstract.** A divisible load is an amount  $W \in \mathbb{R}$  of computational work that can be arbitrarily divided into chunks and distributed among a set P of worker processors to be processed in parallel. Divisible load applications occur in many fields of science and engineering. They can be parallelized in a master-worker fashion, but they pose several scheduling challenges. The Divisible Load Scheduling Problem consists in (a) selecting a subset  $A \subseteq P$  of active workers, (b) defining the order in which the chunks will be transmitted to each of them, and (c) deciding the amount of load  $\alpha_i$  that will be transmitted to each worker  $i \in A$ , with  $\sum_{i \in A} \alpha_i = W$ , so as to minimize the makespan, i.e., the total elapsed time since the master began to send data to the first worker, until the last worker stops its computations. In this work, we propose a biased random-key genetic algorithm for solving the divisible load scheduling problem. Computational results show that the proposed heuristic outperforms the best heuristic in the literature.

**Keywords:** Divisible load scheduling, Random-key genetic algorithms, metaheuristic, parallel processing, scientific computing

<sup>\*</sup> This work was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq), the Foundation for Support of Research of the State of Minas Gerais (FAPEMIG), the Foundation for Support of Research of the State of Rio de Janeiro (FAPERJ), and the Coordination for the Improvement of Higher Education Personnel (CAPES), Brazil

<sup>\*\*</sup> Work of this author was done when he was employed by AT&T Labs Research.

#### 1 Introduction

A divisible load is an amount  $W \ge 0$  of computational work that can be arbitrarily divided and distributed among different processors to be processed in parallel. The processors are arranged in a star topology and the load is stored in a central *master* processor. The master splits the load into chunks of arbitrary sizes and transmits each of them to other *worker* processors. In the remainder of the text we refer to each worker as a processor, while the term master is used to differentiate the master processor from the worker processors.

The master can only send load to one processor at a time. It is assumed that it does not process the load itself. Any processor can only start processing after it has completely received its respective chunk of the load. The processors are heterogeneous in terms of processing power, communication speed, and setup time for start communicating with the master. Not all available processors should necessarily be used for processing the load. Consequently, despite how the load is split, the choice of (i) which processors are used and (ii) the order in which the chunks are transmitted influences the total processing time of the load.

The Divisible Loading Scheduling Problem (DLSP) was introduced in [13], motivated by an application in intelligent sensor networks. Applications of DLSP arise from a number of scientific problems, such as parallel database searching [12], parallel image processing [27], parallel video encoding [28,38], processing of large distributed files [40], and task scheduling in cloud computing [30], among others.

In this work, we deal with the same DLSP variant treated in [1, 11]. Let  $W \in \mathbb{R}$  be the amount of load to be processed, 0 (zero) be the index of the master processor, and  $P = \{1, ..., n\}$  be the set of worker processors indices. Each processor  $i \in P$  has (i) a setup time  $g_i \in \mathbb{R}$  to start the communication with the master, (ii) a communication time  $G_i \in \mathbb{R}$  needed to receive each unit of the load from the master and (iii) a processing time  $w_i \in \mathbb{R}$  needed to process each unit of the load. Therefore, it takes  $g_i + \alpha_i \cdot G_i$  units of time for the master to transmit a load chunk of size  $\alpha_i \in \mathbb{R}$  to the processor  $i \in P$ . Furthermore, it takes an additional  $w_i \cdot \alpha_i$  units of time for this worker to process the chunk of load assigned to it.

The scheduling problem consists of (a) selecting a subset  $A \subseteq P$  of active processors, (b) defining the order in which the chunks will be transmitted to each active processor and (c) deciding the amount of load  $\alpha_i$  that will be transmitted to each processor  $i \in A$ , with  $\sum_{i=1}^{n} \alpha_i = W$ , so as to minimize the makespan, i.e., the total elapsed time since the master began to send data to the first processor, until the last processor stops its computations. This problem was proved NP-Hard in [45]. We note that  $\alpha_i = 0$ , for all  $i \in P \setminus A$ .

An example of an optimal single-round solution for DLSP is displayed in Figure 1, while an example of a non-optimal solution is displayed in Figure 2. One can see three time bars for each processor in these figures. The first bar represents the amount of time that is necessary to start the communication with the master, while the second corresponds to the amount of time needed to receive the respective chunk of load. Finally, the third bar represents the time spent by the worker to process this chunk. It can be seen that the master only starts the communication with a processor after finishing the transmission to the previous one.



Fig. 1. Example of an optimal single-round scheduling.



Fig. 2. Example of a non-optimal single-round scheduling.

Since the load can be split arbitrarily, all processors stop at the same time in the optimal solution [5]. In addition, if the order in which the chunks are transmitted to the processors is fixed, then the best solution can be computed in O(n) time, using the AlgRap algorithm developed in [1]. In other words, given a permutation of the processors in P, AlgRap computes the set of active processors and the amount of load that has to be sent to each of them to minimize the makespan. Therefore, DLSP can be reduced to the problem of finding the best permutation of the processors, i.e., the one which induces the solution with the minimum makespan. In order to find this permutation, we propose a biased random-key genetic algorithm [19, 20, 24], which has been successfully used for solving many permutation based combinatorial optimization problems [18, 20–23, 25, 32, 33].

The remainder of the paper is organized as follows. Related work is reviewed in the next section. The proposed heuristic is described in Section 4. Computational experiments are reported and discussed in Section 5. Concluding remarks are drawn in the last section.

## 2 Related work

There are many variants of DLSP in the literature. Divisible load scheduling may be performed in a *single round* or in *multiple rounds*. In the single-round case [1, 3-5, 7, 9, 11, 14, 16, 17, 26, 29, 35, 39, 41, 42], each active processor receives and processes a single chunk of the load. In the multi-round case [1, 4, 6, 8, 15-17, 34, 36, 42-44], each active processor receives and processes multiple chunks of load. After the master finishes the transmission of the first round of load to all active processors, it immediately starts the transmission of the next batch of load chunks following the same order, until all the load is distributed among the processors.

The processors may be homogeneous or heterogeneous. If the processors are homogeneous, the values of  $g_i$ ,  $G_i$ , and  $w_i$  are the same for all processors  $i \in P$  [4,8,9,26,43,44]. Contrarily, in the heterogeneous case the values of  $g_i$ ,  $G_i$ , and  $w_i$  may be different for each processor [1,3,5,6,11,14–17,29,34–36,39,41–44].

The system can be *dedicated* or *non-dedicated*. When the system is dedicated, it is assumed that all resources (processors, memory, network, etc.) are used to process a single computational load [1,5-7,9,11,14,16,17,34,39,41-44]. Non-dedicated systems may be used to simultaneously process different computational loads [6,7,35].

There may be a limitation to the maximum chunk size that can be received by each processor. When such a limitation does not exist, the problem is said to be *unconstrained* [1,4,5,8,9,11,14,34-36,39,41]. If the maximum chunk size is limited, the problem is said to be *buffer constrained* [6,7,15-17,29,42-44].

In this work, we consider the unconstrained single-round DLSP with dedicated and heterogeneous processors [1, 4, 5, 11, 16, 41, 45], that was proved to be *NP*-hard in [45]. Blazewicz and Drozdowski [11] showed that once a permutation of the processors is given, a solution with minimum makespan can be obtained in  $O(n \log n)$  time, where n = |P| is the number of processors. Later, Abib and Ribeiro [1] proposed the faster AlgRap algorithm that finds this solution in O(n)time. Beaumont et al. [5] showed that if  $g_i = 0$ , for all  $i \in P$ , then DLSP can be polynomially solved by sorting the processors in non-decreasing order of the  $G_i$  values. They also showed that if  $G_i = G_j$ , for all  $i, j \in P$ , then the problem can be solved by sorting the processors in non-decreasing order of the products  $g_i \cdot w_i$ .

Non-linear programming formulations for the unconstrained single-round DLSP with dedicated and heterogeneous processors were proposed in [4, 14]. However,

to the best of our knowledge, there are no efficient algorithms in the literature to solve these formulations. The first mixed integer linear programming formulation for DLSP was proposed in [1]. This formulation is described below and used to assess the quality of the heuristic proposed later in this paper.

Let a DLSP instance be defined by  $\langle W, P, g, G, w \rangle$ . Formulation (1)-(11) rely on decision variables  $x_{ij} \in \{0, 1\}$ , with  $x_{ij} = 1$  if the processor  $i \in P$  is the  $j^{th}$  processor to receive its load, and  $x_{ij} = 0$  otherwise. Variables  $\alpha_{ij} \geq 0$ amount for the size of the load chunk received by the processor  $i \in P$  if it is the  $j^{th}$  processor to receive its load. We notice that  $\alpha_{ij} = 0$  if the processor iis not the  $j^{th}$  to receive its load and that  $\sum_{j \in \{1, \ldots, |P|\}} \alpha_{ij} = 0$  if processor i is not active. Non-negative auxiliary variables T and  $t_j$ , for all  $j \in \{1, \ldots, |P|\}$ , stand for the makespan and the time the  $j^{th}$  processor starts receiving its chunk, respectively:

$$Minimize \quad T \tag{1}$$

$$\sum_{i=1}^{n} x_{ij} \le 1 \qquad \qquad \forall j \in \{1, \dots, n\}$$

$$\tag{2}$$

$$\sum_{j=1}^{n} x_{ij} \le 1 \qquad \qquad \forall i \in \{1, \dots, n\}$$

$$(3)$$

$$\sum_{i=1}^{n} x_{ij} \ge \sum_{i=1}^{n} x_{i,j+1} \qquad \forall j \in \{1, ..., n-1\}$$
(4)

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_{ij} = W \tag{5}$$

$$\alpha_{ij} \le W \cdot x_{ij} \qquad \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\} \qquad (6)$$

$$t_1 = 0 \qquad (7)$$

$$t_j \ge t_{j-1} + \sum_{i=1}^n (g_i \cdot x_{i,j-1} + G_i \cdot \alpha_{i,j-1}) \quad \forall j \in \{2, ..., n\}$$
(8)

$$t_j + \sum_{i=1}^n (g_i \cdot x_{ij} + (G_i + w_i) \cdot \alpha_{ij}) = T \quad \forall j \in \{1, ..., n\}$$
(9)

$$\begin{aligned} x_{ij} \in \{0,1\} & & \forall i \in \{1,...,n\}, \forall j \in \{1,...,n\} \quad (10) \\ \alpha_{ij} \geq 0 & & \forall i \in \{1,...,n\}, \forall j \in \{1,...,n\}. \quad (11) \end{aligned}$$

The objective function (1) consists in minimizing the makespan. Constraints (2) imply that at most one processor can be the  $j^{th}$  to receive data. Constraints (3) ensure that each processor can be activated at most once. Constraints (4) guarantee that there must be j previously activated processors when the  $(j + 1)^{th}$  is activated. Constraint (5) enforces that the full load is divided over the processors. Constraints (6) establish processor i can only be the  $j^{th}$  to receive data if it was chosen to be the  $j^{th}$  in the activation order. Constraint (7) is used

to indicate that data transmission starts at time zero. Constraints (8) are used to enforce that the  $j^{th}$  processor to be activated will start receiving data after the previous processor in the activation order finishes receiving its data. Constraints (9) imply that the makespan T is equal to the time  $t_j$  in which any processor jstarts receiving data plus the time  $\sum_{i=1}^{n} g_i x_{ij} + (G_i + w_i)\alpha_{ij}$  it needs to receive and process it. These constraints rely on the fact that, in any optimal solution, all processors finish at the same time [10]. Constraints (10) and (11) define the integrality of variables  $x_{ij}$  and the non-negativity of variables  $\alpha_{ij}$ , respectively. This formulation improves and extends that in [4].

Computational experiments reported in [1] have shown that the CPLEX branch-and-cut algorithm based on this formulation was able to find optimal solutions for 490 out of 720 instances with up to 160 processors. However, for the largest unsolved instances, the integrality gaps were very high, which motivated the development of heuristics for solving DLSP.

Once again, to the best of our knowledge, the HeuRet heuristic proposed by Abib and Ribeiro [1] for the DLSP variant studied in this paper is the most effective in the literature. At each iteration, their algorithm (i) estimates a performance index  $e_i$  for each processor  $i \in P$  and (ii) builds a solution by taking the processors in P in a non-increasing order of the  $e_i$  values. The algorithm sets  $e_i = G_i$ , for all  $i \in P$ , in the first iteration and makes use of the exact algorithm AlgRap to compute an initial solution  $s_0$  with makespan  $T_0$ . Next, and at each forthcoming iteration k, the estimated performance  $e_i$  of each processor  $i \in P$ is updated using the values of  $g_i$ ,  $w_i$ ,  $G_i$ , and  $T_{k-1}$  and a new solution with makespan  $T_k$  is built by algorithm AlgRap considering the new order defined on the processors by the newly updated performance indices  $e_i$ . The procedure stops when  $T_{k-1} < T_k$ , i.e., when the new solution degenerates the makespan of the previous one due to the use of some processor that is not needed.

The heuristic proposed in Section 4 improves upon HeuRet because, instead of defining a greedy local search based on the performance estimations of the processors, it performs a global search in the space of processor permutations in order to find the permutation that induces the optimal or a near-optimal solution. As both HeuRet and the new heuristic rely on the AlgRap algorithm, we describe it in the next section.

## 3 Solving DLSP with a fixed activation order

In this section, we describe the linear-time algorithm AlgRap proposed in [1] for the special case in which the processor activation order is fixed beforehand. In this case, the scheduling problem consists exclusively in computing the load to be sent to each processor.

Without loss of generality and for easiness of notation, we assume that processor  $i \in P$  is the *i*-th to be activated. We denote by  $\alpha_i^*$  the optimal amount of data to be sent to processor *i* and we define  $f_i = (w_i + G_i)/w_{i-1}$ , for  $i = 1, \ldots, n$ .

#### 3.1 Feasible solutions with a given number of processors

We first suppose that the number  $\ell$  of processors to be used is also known. In this case, Blazewicz and Drozdowski [11] established that the solution to the system

$$\alpha_k \cdot w_k = g_{k+1} + \alpha_{k+1} \cdot (w_{k+1} + G_{k+1}), \quad k = 1, \dots, \ell - 1$$
(12)

$$\sum_{k=1}^{\ell} \alpha_k = W \tag{13}$$

gives the optimal loads:

$$\alpha_k = \alpha_\ell \prod_{j=k+1}^\ell f_j + \sum_{j=k+1}^\ell \left(\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i\right), \quad k = 1, \dots, \ell - 1$$
(14)

$$\alpha_{\ell} = \frac{W - \sum_{k=1}^{\ell-1} \sum_{j=k+1}^{\ell} \left(\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i\right)}{1 + \sum_{k=1}^{\ell-1} \prod_{j=k+1}^{\ell} f_j}.$$
(15)

These values yield a feasible solution if  $\alpha_k \geq 0$ , for every  $k = 1, \ldots, \ell$ . Equations (12) imply that the products  $\alpha_i \cdot w_i$  are non-increasing for  $i = 1, \ldots, \ell$ , since all constants  $G_i$ ,  $w_i$ , and  $g_i$  are non-negative. Therefore,  $\alpha_i \geq 0$  if and only if  $\alpha_\ell \geq 0$ , for  $i = 1, \ldots, \ell$ . Finally, we conclude from equation (15) that the above solution is feasible if and only if  $V(\ell) = \sum_{k=1}^{\ell-1} \sum_{j=k+1}^{\ell} \left(\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i\right) \leq W$ .

#### 3.2 Optimal number of processors

We now investigate the exact number  $\ell^*$  of processors that should be activated in an optimal solution. The term  $V(\ell)$  appearing in the numerator of equation (15) may be recursively defined by

$$V(\ell) = \frac{g_{\ell}}{w_{\ell-1}} \sum_{k=1}^{\ell-1} \prod_{i=k+1}^{\ell-1} f_i + V(\ell-1).$$
(16)

For any activation order,  $V(\ell)$  is a non-decreasing function of the number  $\ell$  of activated processors. A solution is infeasible if  $V(\ell) > W$ . Let the function  $F(\ell) = \sum_{k=1}^{\ell-1} \prod_{i=k+1}^{\ell-1} f_i$  be recursively defined by

$$F(1) = 0,$$
  
 $F(2) = 1,$  and  
 $F(\ell) = 1 + F(\ell - 1)f_{\ell-1}$ 

Therefore, V(1) = 0 and for any  $\ell > 1$ 

$$V(\ell) = \frac{g_{\ell}}{w_{\ell-1}} F(\ell) + V(\ell-1)$$
(17)

may be computed in time O(1) from  $V(\ell - 1)$  and  $F(\ell)$ .

Let us assume that a feasible solution exists for r processors and suppose that one additional processor is made available. If a feasible solution satisfying constraints (12-13) still exists, then some load will be transferred from one of the original r processors to the new processor taking part in the computations. In consequence, the loads assigned to the other processors will be decreased and the makespan will be reduced. Therefore, the optimal solution (i.e., that with minimum makespan) will have as many processors as possible to achieve feasibility.

#### 3.3 Linear-time algorithm

Given a fixed activation order, the algorithm starts by sending all the load exclusively to the first processor. Next, the number  $\ell$  of processors is iteratively increased from 1 to n, until  $V(\ell)$  turns out to be greater than W. Then, the optimal number of processors is set as  $\ell^* = \ell - 1$ .

Once the optimal number  $\ell^*$  of processors has been computed, the load  $\alpha_{\ell^*}$  sent to the last processor is computed from equation (15). The other loads  $\alpha_i$ , for  $i = 1, \ldots, \ell^* - 1$ , are recursively computed from  $\ell^*$  using equation (14). Algorithm 1 implements the above computations in time O(n).

In addition to the number of processors and all their data, this algorithm takes as input a vector  $\pi$  describing the activation order, such that  $\pi(i) = j$  indicates that processor j is the *i*-th to be activated, for  $i, j = 1, \ldots, n$ . For instance, if n = 3 and  $\pi = \langle 2, 3, 1 \rangle$ , then processor 2 is the first to be activated, processor 3 is the second, and processor 1 is the third.

## 4 Biased random-key genetic algorithm

Genetic algorithms with random keys, or random-key genetic algorithms (RKGA), were first introduced by [2] for combinatorial optimization problems for which solutions can be represented as a permutation vector. 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 a RKGA. Parents are allowed to be selected for mating more than once in a given generation.

A biased random-key genetic algorithm (BRKGA) differs from a RKGA in the way parents are selected for crossover, see Gonçalves and Resende [20] 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 biased because one parent is always an elite individual and has a higher probability of passing its genes to the new generation.

**Require:** Vector  $\pi$  establishing the activation order **Ensure:** Minimum makespan  $T^*$ , number  $\ell^*$  of processors, loads  $\alpha_i$  for  $i = 1, \ldots, n$ 1:  $F[1] \leftarrow 0$ 2:  $F[2] \leftarrow 1$ 3: for i = 3 to n do 4:  $F[i] \leftarrow 1 + F[i-1] \cdot (w_{\pi[i-1]} + G_{\pi[i-1]}) / w_{\pi[i-2]}$ 5: end for 6:  $V[1] \leftarrow 0$ 7: for i = 2 to n do 8:  $V[i] \leftarrow (g_{\pi[i]}/w_{\pi[i-1]}) \cdot F[i] + V[i-1]$ 9: end for 10: for  $\ell = 1$  to n do 11: if  $V[\ell] \leq W$  then  $\ell^* \leftarrow \ell$ 12:13: end if 14: end for 15:  $numerator \leftarrow W - V[\ell^*]$ 16:  $product \leftarrow (w_{\pi[\ell^*]} + G_{\pi[\ell^*]})/w_{\pi[\ell^*-1]}$ 17:  $denominator \leftarrow 1$ 18: for  $k = \ell^* - 1$  down to 1 do  $19: \quad denominator \leftarrow denominator + product$ 20: if  $k \neq 1$  then 21:  $product \leftarrow product \cdot (w_{\pi[k]} + G_{\pi[k]})/w_{\pi[k-1]}$ 22: end if 23: end for 24:  $\alpha_{\pi[\ell^*]} \leftarrow numerator/denominator$ 25: for  $k = \ell^* - 1$  to 1 do 26:  $\alpha_{\pi[k]} \leftarrow (g_{\pi[k+1]} + \alpha_{\pi[k+1]} \cdot (w_{\pi[k+1]} + G_{\pi[k+1]}))/w_{\pi[k]}$ 27: end for 28: for  $k=\ell^*+1$  to n do 29:  $\alpha_{\pi[k]} \leftarrow 0$  $30:\ \mathbf{end}\ \mathbf{for}$ 31:  $T^* \leftarrow \alpha_{\pi[1]} \cdot (w_{\pi[1]} + G_{\pi[1]}) + g_{\pi[1]}$  ${\bf Algorithm}~{\bf 1}:$  Linear-time algorithm AlgRap for a fixed activation order.

The BRKGA-DLS biased random-key genetic algorithm for DLSP evolves a population of chromosomes that consists of vectors of real numbers (keys). Each solution is represented by a |P|-vector, in which each component is a real number in the range [0, 1] associated with a processor in P. Each solution represented by a chromosome is decoded by a heuristic that receives the vector of keys and builds a feasible solution for DLSP. The decoding heuristic is based on algorithm AlgRap described in the previous section. Decoding consists of two steps: first, the processors are sorted in a non-decreasing order of their random keys; next, the resulting order is used as the input for AlgRap. The makespan of the solution provided by AlgRap is used as the fitness of the chromosome.

We use the parametric uniform crossover scheme proposed in [37] to combine two parent solutions and produce an offspring. In this scheme, the offspring inherits each of its keys from the best fit of the two parents with probability 0.60 and from the least fit parent with probability 0.40. This genetic algorithm 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: a fixed number of 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.

The keys associated to each processor are randomly generated in the initial population. At each generation, the population is partitioned into two sets: TOP and REST. Consequently, 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 on the current population. As illustrated in Figure 3, 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 [2], 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. In our implementation, the population size was set to  $|TOP| + |MID| + |BOT| = 5 \times |P|$ , with the sizes of sets TOP, MID, and BOT set to  $0.15 \times 5 \times |P|$ ,  $0.7 \times 5 \times |P|$ , and  $0.15 \times 5 \times |P|$ , respectively, as suggested by Noronha et al. [32].

## 5 Computational experiments

In this section, we report computational experiments to assess the performance of the biased random-key genetic algorithm BRKGA-DLS. This algorithm was



Fig. 3. Population evolution between consecutive generations of a BRKGA.

implemented in C++ and compiled with GNU C++ version 4.6.3. The experiments were performed on a Quad-Core AMD Opteron(tm) Processor 2350, with 16 GB of RAM memory.

The set of instances used in the three first experiments was the same proposed in [1]. There are 120 grid configurations with n = 10, 20, 40, 80, 160 worker processors and eight combinations of the parameter values  $g_i$ ,  $G_i$  and  $w_i$ ,  $i = 1, \ldots, n$ , each of them ranging either in the interval [1, 100], denoted as *low*, or in the interval [1000, 100,000], denoted as *high*. Three instances were generated for each combination of  $n, g_i, G_i$ , and  $w_i$ . Six different values of the load W = 100, 200, 400, 800, 1600, 3200 are considered for each grid configuration, corresponding to 18 instances for each combination of parameters  $g_i, G_i$  and  $w_i$ , amounting to a total of 720 test instances. Each heuristic was run 10 times for each instance, with different seeds for the random number generator of [31].

The first experiment consisted in evaluating if BRKGA-DLS efficiently identifies the relationships between keys and good solutions and converges faster to near-optimal solutions. We compare its performance with that of a multi-start procedure that uses the same decoding heuristic as BRKGA-DLS. Each iteration of the multi-start procedure, called MS-DLS, applies the same decoding heuristic of BRKGA-DLS, but using randomly generated values for the keys. In this experiment, BRKGA-DLS was run for 1000 generations and MS-DLS for  $1000 \times q$  iterations, where  $q = 5 \times |P|$  is the population size of BRKGA-DLS. The results are reported in Figure 4. The first bar gives the average over the 720 instances of the percent relative reduction between the worst solution value returned by BRKGA-DLS with respect to that obtained by MS-DLS. The next two bars give the same information for the average and best solution values. It can be seen that the average solution values found by BRKGA-DLS were 4.95% better than those provided by MS-DLS. Also, the worst (resp. best) solution values found by BRKGA-DLS were 5.98% (resp. 3.78) smaller than the respective worst (resp. best) solution values obtained with MS-DLS. These results indicate that BRKGA-DLS identifies the relationships between keys and good solutions, making the evolutionary process converge to better solutions faster than MS-DLS.



Fig. 4. Average percent relative reduction over the 720 instances of the best, average and worse solution values found by BRKGA-DLS with respect to those obtained by MS-DLS.

In the second experiment, we compare BRKGA-DLS with HeuRet and MS-DLS. We first evaluated how many optimal solutions have been obtained by each heuristic over the 720 test instances. The default CPLEX branch-and-cut algorithm based on the formulation of [1] was also run for the 720 instances. Version 12.6 of CPLEX was used and the maximum CPU time was set to 24 hours. CPLEX was able to prove the optimality for 497 out of the 720 test instances. The first line of Table 1 shows that BRKGA-DLS found optimal solutions for 413 instances (i.e. 83.1%) out of the 497 instances for which the optimal solutions are known, while HeuRet found 320 optimal solutions and MS-DLS found only 177 of them. The second line of Table 1 gives the number of instances for which each heuristic found the best known solution value. The third line of this table shows the number of runs for which BRKGA-DLS and MS-DLS found the best known solution values (we recall that HeuRet is a deterministic algorithm, while the others are randomized). Finally, the last line of this table gives, for each of the three heuristics, a score that represents the sum over all instances of the number of methods that found strictly better solutions than the specific heuristic being considered. The lower a score is, the best the corresponding heuristic is. It can be seen that BRKGA-DLS outperformed both HeuRet and MS-DLS heuristics with respect to the number of optimal and best solutions found, as well as with respect to the score value. In particular, BRKGA-DLS obtained better scores

than HeuRet for all but one instance and found the best known solution values for 645 out of the 720 test instances, while HeuRet found the best solution values for only 313 instances.

**Table 1.** Summary of the numerical results obtained with BRKGA-DLS, HeuRet andMS-DLS for 720 test instances.

	MS-DLS	HeuRet	BRKGA-DLS
Optimal values (over 497 instances)	177	320	413
Best values (over 720 instances)	189	313	645
Best values (over 7200 runs)	2166	-	6191
Score value	803	112	1

In the third experiment, we assess the computation times of BRKGA-DLS and show how they grow with the number of processors. The results are shown in Table 2. The three first columns show the ranges of values of  $w_i$ ,  $g_i$ , and  $G_i$ , respectively, for each group of instances. The five next columns display the results for the instances with n equal to 10, 20, 40, 80, and 160. Since there are three instances for each of the six values of W, each table cell gives the average CPU time of BRKGA-DLS over 18 instances with the same values of  $w_i$ ,  $g_i$ ,  $G_i$ , and n. One can see that the average CPU time for the instances with 10, 20, 40, 80, and 160 was, respectively, 0.07, 0.31, 1.31, 5.63, 25.18 seconds. These results show that the average computation time of BRKGA-DLS increases linearly with the number of processors by a factor of approximately four every time the number of processors is doubled. The maximum average computation time observed for BRKGA-DLS was 25.18 seconds for the instances with 160 processors.

$w_i$	$g_i$	$G_i$	10	20	40	80	160
low	low	low	0.07	0.30	1.28	5.50	24.65
low	low	high	0.06	0.29	1.24	5.40	24.40
low	high	low	0.06	0.29	1.23	5.41	24.34
low	high	high	0.06	0.29	1.23	5.38	24.43
high	low	low	0.08	0.34	1.48	6.48	28.87
high	low	high	0.08	0.35	1.35	5.70	25.23
high	high	low	0.08	0.33	1.36	5.64	24.84
high	high	high	0.07	0.30	1.29	5.51	24.69
Avera	ge		0.07	0.31	1.31	5.63	25.18

Table 2. Running times in seconds for BRKGA-DLS.

The fourth and last experiment provides a more detailed comparison between HeuRet and BRKGA-DLS, based on 20 new, larger and more realistic instances with |P| = 320 and W = 10.000. The values of  $G_i$  and  $g_i$  have been

randomly generated in the ranges [1, 100] and [100, 100.000], respectively. However, differently from [1], the values of  $w_i$  have been randomly generated in the interval [200, 500]. These values are more realistic, since the processing rate of a real computer is always larger than its communication rate. In this experiment, BRKGA-DLS was made to stop after |P| generations without improvement in the best solution found. The results are reported in Table 3. The first column shows the instance name. The second and third columns display, respectively, the makespan and the CPU time (in seconds) obtained by HeuRet. The next two columns provide the average makespan over ten runs of BRKGA, the corresponding coefficient of variation, defined as the ratio of the standard deviation to the average. The average computation time in seconds of BRKGA over ten runs is given in the sixth column. The last column shows the percent relative reduction between the average solution found by BRKGA-DLS with respect to that found by HeuRet. It can be seen that the average makespan obtained by BRKGA-DLS is always smaller than that given by HeuRet. In addition, the coefficient of variation of BRKGA-DLS is very small, indicating that it is a robust heuristic. The percent relative reduction of BRKGA-DLS with respect to HeuRet amounted to 3.19% for instance dls.320.10 and to 2.38% on average. As in reallife applications the load size may be very large, and the total communication and processing times may take many hours, an average reduction of 2.38% may be very significant. Although the running times of BRKGA-DLS are larger than those of HeuRet, their average values never exceeded the time taken by HeuRet by more than 30 seconds. Since practical applications of parallel processing take long elapsed times, the trade-off between the reduction in the elapsed time and this small additional running time needed by BRKGA accounts very favorably to BRKGA-DLS.

#### 6 Conclusions

We considered the unconstrained single-round divisible load scheduling problem with dedicated and heterogeneous processors. A new heuristic biased random-key genetic algorithm has been proposed for the problem.

The BRKGA-DLS heuristic improves upon the best heuristic in the literature in terms of solution quality, since it performs a global search in the space of processor permutations in order to find the best activation sequence of the processors.

Computational experiments on 720 test instances with up to 160 processors have shown that BRKGA-DLS found optimal solutions for 413 instances (out of the 497 instances where the optimal solution is known), while the HeuRet heuristic found optimal solutions for only 320 of them. Moreover, BRKGA-DLS obtained better scores than HeuRet for all but one instance and found solutions as good as the best known solution for 645 out of the 720 test instances. To summarize, BRKGA-DLS outperformed the previously existing HeuRet heuristic with respect to all measures considered in Table 1.

	Heu	$_{1}$ Ret		BRKGA		
Instance	makespan	time (s)	makespan	CV (%)	time (s)	reduction (%)
dls.320.01	312813.64	0.01	306613.22	0.04	27.24	1.98
dls.320.02	321764.07	0.01	313847.15	0.07	16.72	2.46
dls.320.03	402264.85	0.01	392059.46	0.11	23.72	2.54
dls.320.04	348474.15	0.01	341436.89	0.02	28.16	2.02
dls.320.05	342086.46	0.01	334946.37	0.03	21.67	2.09
dls.320.06	311824.17	0.01	305601.28	0.02	21.93	2.00
dls.320.07	325732.30	0.01	316467.42	0.02	23.19	2.84
dls.320.08	323171.95	0.01	315065.11	0.03	26.23	2.51
dls.320.09	312326.77	0.01	305948.81	0.02	25.03	2.04
dls.320.10	296984.47	0.01	287521.34	0.12	24.04	3.19
dls.320.11	290559.21	0.01	284822.15	0.04	20.56	1.97
dls.320.12	343076.56	0.01	333085.72	0.05	19.53	2.91
dls.320.13	287276.21	0.01	281525.27	0.04	22.77	2.00
dls.320.14	311054.47	0.01	303796.42	0.06	26.81	2.33
dls.320.15	362369.67	0.01	352642.18	0.04	20.41	2.68
dls.320.16	287083.60	0.01	281082.29	0.09	25.38	2.09
dls.320.17	339666.43	0.01	329893.61	0.04	23.53	2.88
dls.320.18	368795.14	0.01	361281.06	0.07	22.08	2.04
dls.320.19	347671.73	0.01	338075.70	0.03	27.59	2.76
dls.320.20	372427.24	0.01	364013.40	0.06	18.28	2.26
Average	330371.15	0.01	322486.24	0.05	23.24	2.38

Table 3. BRKGA vs. HeuRet on the largest instances with 320 processors.

For the new set of larger and more realistic instances with 320 processors, BRKGA-DLS found solution values 2.38% better than HeuRet on average. In addition, the processing times of BRKGA-DLS are relatively small and never exceeded 30 seconds. Therefore, parallel processing applications dealing with large amounts of data and taking long elapsed times can benefit from BRKGA-DLS, since the additional running time needed by BRKGA-DLS may result in a significant reduction in the makespan.

We are currently working on the extension of this approach to the harder case of multi-round (or multi-installment) scheduling. In this case, the load is distributed to the active processors in several consecutive bursts, reducing the waste in each processor and making better use of the resources to reduce the overall makespan, as illustrated in Figure 5.

# References

- 1. 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, pp. 54–61. Nashville (2009)
- Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 2, 154–160 (1994)



Fig. 5. Example of a multi-round scheduling.

- Beaumont, O., Bonichon, N., Eyraud-Dubois, L.: Scheduling divisible workloads on heterogeneous platforms under bounded multi-port model. In: International Symposium on Parallel and Distributed Processing, pp. 1–7. IEEE, Miami (2008)
- Beaumont, O., Casanova, H., Legrand, A., Robert, Y., Yang, Y.: Scheduling divisible loads on star and tree networks: results and open problems. IEEE Transactions on Parallel and Distributed Systems 16, 207–218 (2005)
- Beaumont, O., Legrand, A., Robert, Y.: Optimal algorithms for scheduling divisible workloads on heterogeneous systems. In: 12th Heterogeneous Computing Workshop, pp. 98–111. IEEE Computer Society Press, Nice (2003)
- Berlinska, J., Drozdowski, M.: Heuristics for multi-round divisible loads scheduling with limited memory. Parallel Computing 36, 199–211 (2010)
- Berlińska, J., Drozdowski, M., Lawenda, M.: Experimental study of scheduling with memory constraints using hybrid methods. Journal of Computational and Applied Mathematics 232, 638–654 (2009)
- Bharadwaj, V., Ghose, D., Mani, V.: Multi-installment load distribution in tree networks with delays. IEEE Transactions on Aerospace and Electronic Systems 31, 555–567 (1995)
- Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.G.: Scheduling divisible loads in parallel and distributed systems. Wiley - IEEE Computer Society Press (1996)
- Blazewicz, J., Drozdowski, M.: Scheduling divisible jobs on hypercubes. Parallel Computing 21, 1945–1956 (1995)
- Blazewicz, J., Drozdowski, M.: Distributed processing of divisible jobs with communication startup costs. Discrete Applied Mathematics 76, 21–41 (1997)
- Błażewicz, J., Drozdowski, M., Markiewicz, M.: Divisible task scheduling-concept and verification. Parallel Computing 25, 87–98 (1999)
- Cheng, Y.C., Robertazzi, T.G.: Distributed computation with communication delay. IEEE Transactions on Aerospace and Electronic Systems 24, 700–712 (1988)
- Drozdowski, M.: Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems. 321. Politechnika Poznanska (1997)
- Drozdowski, M., Lawenda, M.: Multi-installment divisible load processing in heterogeneous systems with limited memory. Parallel Processing and Applied Mathematics **3911**, 847–854 (2006)
- Drozdowski, M., Wolniewicz, P.: Divisible load scheduling in systems with limited memory. Cluster Computing 6, 19–29 (2003)

- Drozdowski, M., Wolniewicz, P.: Optimum divisible load scheduling on heterogeneous stars with limited memory. European Journal of Operational Research 172, 545–559 (2006)
- Duarte, A., Mart, R., Resende, M., Silva, R.: Improved heuristics for the regenerator location problem. International Transactions in Operational Research 21, 541–558 (2014)
- 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, 299– 333 (2002)
- Gonçalves, J.F., Resende, M.G.: Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics 17, 487–525 (2011)
- Gonçalves, J.F., Resende, M.G.: A biased random key genetic algorithm for 2D and 3D bin packing problems. International Journal of Production Economics 145, 500–510 (2013)
- Gonçalves, J.F., Resende, M.G.: An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. International Transactions in Operational Research 21, 215–246 (2014)
- Gonçalves, J.F., Resende, M.G., Toso, R.F.: Biased and unbiased random-key genetic algorithms: An experimental analysis. Tech. rep., AT&T Labs Research, Florham Park (2012)
- Gonçalves, J.F., Resende, M.G.C.: An evolutionary algorithm for manufacturing cell formation. Computers and Industrial Engineering 47, 247–273 (2004)
- Gonçalves, J.F., Resende, M.G.C., Costa, M.D.: A biased random-key genetic algorithm for the minimization of open stacks problem. International Transactions in Operational Research (2014). DOI 10.1111/itor.12109
- Kim, H.J.: A novel optimal load distribution algorithm for divisible loads. Cluster Computing- The Journal of Networks Software Tools and Applications 6, 41–46 (2003)
- Lee, C.k., Hamdi, M.: Parallel image processing applications on a network of workstations. Parallel Computing 21, 137–160 (1995)
- Li, P., Veeravalli, B., Kassim, A.A.: Design and implementation of parallel video encoding strategies using divisible load analysis. IEEE Transactions on Circuits and Systems for Video Technology 15, 1098–1112 (2005)
- Li, X., Bharadwaj, V., Ko, C.: Divisible load scheduling on single-level tree networks with buffer constraints. IEEE Transactions on Aerospace and Electronic Systems 36, 1298–1308 (2000)
- Lin, W., Liang, C., Wang, J.Z., Buyya, R.: Bandwidth-aware divisible task scheduling for cloud computing. Software: Practice and Experience 44, 163–174 (2014)
- Matsumoto, M., Nishimura, T.: Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation 8, 3–30 (1998)
- Noronha, T.F., Resende, M.G.C., Ribeiro, C.C.: A biased random-key genetic algorithm for routing and wavelength assignment. Journal of Global Optimization 50, 503–518 (2011)
- Resende, M.G.: Biased random-key genetic algorithms with applications in telecommunications. TOP 20, 130–153 (2012)
- 34. Shokripour, A., Othman, M., Ibrahim, H.: A method for scheduling last installment in a heterogeneous multi-installment system. In: Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology, pp. 714–718. Chengdu (2010)

- Shokripour, A., Othman, M., Ibrahim, H., Subramaniam, S.: A new method for job scheduling in a non-dedicated heterogeneous system. Procedia Computer Science 3, 271–275 (2011)
- Shokripour, A., Othman, M., Ibrahim, H., Subramaniam, S.: New method for scheduling heterogeneous multi-installment systems. Future Generation Computer Systems 28, 1205–1216 (2012)
- Spears, W., deJong, K.: 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. Morgan Kaufman, San Mateo (1991)
- Turgay, A., Yakup, P.: Optimal scheduling algorithms for communication constrained parallel processing. Lecture Notes in Computer Science 2400, 197–206 (2002)
- 39. Wang, M., Wang, X., Meng, K., Wang, Y.: New model and genetic algorithm for divisible load scheduling in heterogeneous distributed systems. International Journal of Pattern Recognition and Artificial Intelligence 27 (2013)
- Wang, R., Krishnamurthy, A., Martin, R., Anderson, T., Culler, D.: Modeling communication pipeline latency. ACM Sigmetrics Performance Evaluation Review 26, 22–32 (1998)
- Wang, X., Wang, Y., Meng, K.: Optimization algorithm for divisible load scheduling on heterogeneous star networks. Journal of Software 9, 1757–1766 (2014)
- 42. Wolniewicz, P.: Divisible job scheduling in systems with limited memory. Ph.D. thesis, Poznan University of Technology, Poznań (2003)
- 43. Yang, Y., Casanova, H.: RUMR: Robust scheduling for divisible workloads. In: Proceedings of the 12th IEEE Symposium on High Performance and Distributed Computing, pp. 114–125. Seattle (2003)
- 44. Yang, Y., Casanova, H.: UMR: A multi-round algorithm for scheduling divisible workloads. In: Proceedings of the 17th International Parallel and Distributed Processing Symposium, pp. 24–32. Nice (2003)
- 45. Yang, Y., Casanova, H., Drozdowski, M., Lawenda, M., Legrand, A., et al.: On the complexity of multi-round divisible load scheduling. Tech. Rep. 6096, INRIA Rhône-Alpes (2007)

ANEXO D – 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". A ser submetido para Journal of Scheduling.

# A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems

Julliany S. Brandão<sup>1</sup>,<sup>2</sup> \*, Thiago F. Noronha<sup>3\*</sup>, Mauricio G. C. Resende<sup>4\*\*</sup>, and Celso C. Ribeiro<sup>1\*</sup>

 <sup>1</sup> Universidade Federal Fluminense Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil {jbrandao,celso}@ic.uff.br
 <sup>2</sup> Centro Federal de Educação Tecnológica Celso Suckow da Fonseca Av. Maracanã, 229, Rio de Janeiro, RJ 20271-110, Brazil
 <sup>3</sup> Universidade Federal de Minas Gerais Avenida Antônio Carlos 6627, Belo Horizonte, MG 24105, Brazil tfn@dcc.ufmg.br
 <sup>4</sup> Mathematical Optimization and Planning, Amazon.com 333 Boren Avenue North, Seattle, WA 98109, USA resendem@amazon.com

**Abstract.** A divisible load is an amount W of computational work that can be arbitrarily divided into independent chunks. In many divisible load applications, the load can be parallelized in a master-worker fashion, where the master distributes the chunks among a set P of worker processors to be processed in parallel. The master can only send load to one worker at a time, and the transmission can be done in a single round or in multiple rounds. The multi-round divisible load scheduling problem consists in (a) selecting the subset  $A \subseteq P$  of workers that will process the load, (b) defining the order in which the chunks will be transmitted to each of them, (c) defining the number m of transmission rounds that will be used, and (d) deciding the amount of load that will be transmitted for each worker  $i \in A$  at each round  $j \in \{1, \ldots, m\}$ , so as to minimize the makespan. The state-of-the-art heuristic in the literature finds a feasible solution using closed-form formulas. In this work, we propose an alternative heuristic approach that decides the transmission order, the set Aand the value m by means of a biased random-key genetic algorithm. Computational results showed that the proposed heuristic outperforms the previous algorithm algorithm by 11.68%, on average.

**Keywords:** Divisible load scheduling, Multi-round, Random-key genetic algorithms, metaheuristic

<sup>&</sup>lt;sup>\*</sup> This work was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq), the Foundation for Support of Research of the State of Minas Gerais (FAPEMIG), the Foundation for Support of Research of the State of Rio de Janeiro (FAPERJ), and the Coordination for the Improvement of Higher Education Personnel (CAPES), Brazil.

 $<sup>^{\</sup>star\star}$  Work of this author was done when he was employed by AT&T Labs Research.

# 1 Introduction

A divisible load is an amount W of computational work that can be arbitrarily divided and distributed among different processors to be computed in parallel. The processors are arranged in a star topology and the load is stored in a central master processor. The master splits the load into chunks of arbitrary sizes and transmits each of them to the worker processors. It is assumed that the master can only send load to one worker at a time, and that it does not compute the load itself. Besides, a worker can only start computing a load chunk after its transmission is finished.

The workers are heterogeneous in terms of processing power, communication speed, and setup times. Due to communication and setup times, it might be faster not to use all the available worker processors to compute the load. Those used in this computation are referred to as *active workers*. The order the master uses to send the load for each worker is called the *transmission order*. Once the latter is set, the transmission can be performed in a single round or in multiple rounds.

In single-round systems [1, 5, 6, 8, 10, 12, 13, 18, 21, 22, 40, 41, 52, 55-57], the active workers receive their piece of the load in a single chunk, and the setup costs are paid only once for each worker. However, in this case, the *i*-th worker in the transmission order has to wait idle until the master transmits all the load chunks to all previous workers. An example of an optimal single-round schedule is shown in Figure 1. All the ten workers are active and the transmission order is < 6, 1, 8, 7, 9, 5, 3, 10, 4, 2 >. Black boxes represent the setup time to start the communication with the master, dashed boxes correspond to the amount of time needed to receive the respective load chunk, and gray boxes show the time spent by the worker to process this chunk. This example does not show setup the times to start processing the load. It can be seen that the master only starts the communication with a worker after it finishes the communication with the previous worker in the transmission order. One can also see that each worker receives a single load chunk and that it only starts processing this chunk after its transmission is finished. As the load can be split arbitrarily, all processors stop at the same time in the optimal solution [8].

In multi-round systems [6,7,9,11,19,20,39,42,43,51,53,57-61], the transmission is divided into rounds and every active worker receives a load chunk at each round. The transmission order is the same for all rounds [58,59,61]. Multi-round systems may reduce the makespan by decreasing the idle times, at the additional cost of repeated setup times at each round. Figure 2 illustrates an optimal three-round schedule. The transmission order is < 6, 8, 7, 1, 9, 3, 5, 2, 4, 10 >, and the first seven workers are active. It can be seen that the master only starts the communication with a worker after it finishes the communication with the previous worker in the activation order. However, idle times in the first round are smaller, because the load chunks are spread among the three rounds. One can also see that each worker only starts processing a load chunk after its transmission is finished. Again, as the load can be split arbitrarily, all processors stop at the same time in the optimal solution [8].



Fig. 1. Example of an optimal single-round scheduling.



Fig. 2. Example of an optimal three-round scheduling.

We focus on the multi-round scheduling systems introduced by [8,53]. Let  $W \in \mathbb{R}_+$  the amount of load to be processed, 0 (zero) be the index of the master processor, and  $P = \{1, ..., |P|\}$  be the set of worker processors indices. Each processor  $i \in P$  has (i) a setup time  $g_i \in \mathbb{R}_+$  to start the communication with the master, (ii) a communication time  $G_i \in \mathbb{R}_+$  needed to receive each unit of the load from the master, (iii) a setup time  $s_i \in \mathbb{R}_+$  to start the load computation and (iv) a processing time  $w_i \in \mathbb{R}_+$  needed to process each unit of the load. Therefore, it takes  $g_i + G_i \cdot \theta_i^j$  units of time for the master to transmit a load chunk of size  $\theta_i^j \in \mathbb{R}_+$  to the processor  $i \in P$  in the round j. Furthermore, it takes an additional  $s_i + w_i \cdot \theta_i^j$  units of time for this worker to process the chunk of load assigned to it.

The Multi-Round Divisible Loading Scheduling problem (MR-DLS) consists in (a) defining the transmission order  $\pi$ , (b) deciding the number n of active workers, (c) choosing the number m of rounds, and (d) deciding the amount of load  $\theta_i^j$  that will be transmitted to each processor  $i \in P$  at a round  $j \in$  $\{1, \ldots, m\}$ , with  $\sum_{i \in P} \sum_{j \in \{1, \ldots, m\}} \theta_i^j = W$ , so as to minimize the makespan. The latter is defined as the total elapsed time since the master began to send data to the first worker, until the last worker stops its computation. Without loss of generality, and for easiness of notation, we assume that processor  $i \in P$  is the *i*-th worker in  $\pi$  in the remainder of this manuscript. Besides we assume that the first n workers in  $\pi$  are active, i.e.  $A = \{i | i = 1 \dots n\}$ , and that  $\theta_i^j = 0$ , for all  $i \in \{n + 1, \dots | P |\}$  and  $j \in \{1, \dots, m\}$ .

The state-of-the-art approach in the literature of MR-DLS was proposed in [53]. It uses a fixed transmission order and applies closed-form formulas to decide the values of n and m. In this work, we propose an alternative heuristic approach for MR-DLS based on the biased random-key genetic algorithm [24, 29, 35]. The latter has been successfully used for solving many permutation based combinatorial optimization problems [23, 29–31, 34, 36, 45, 46], including the single-round divisible load scheduling problem [14]. Computational results showed that the proposed heuristic outperforms the best algorithm by 11.68%, on average.

The remainder of the paper is organized as follows. Related works are presented in the next section. The proposed heuristic is described in Section 3. Computational experiments are reported and discussed in Section 4, while concluding remarks are drawn in the last section.

# 2 Related work

There are many variants of the Divisible Load Scheduling (DLS) problem. The main differences between these variants are discussed below, followed by a literature review on the particular variant studied in this paper.

There are DLS variants where the worker processors are homogeneous and others where they are heterogeneous. In the former case, the values of  $g_i$ ,  $G_i$ ,  $s_i$  and  $w_i$  are the same for all processors  $i \in P$  [6, 11, 12, 40, 42, 43, 58, 59, 61]. In the latter case, these values may be different for each worker [1, 5, 7–9, 13, 18–22, 39, 41, 51–53, 55–61]. In practice, heterogeneous systems are more common than homogeneous systems.

The model of the system adopted can *dedicated* or *non-dedicated*. When the system is *dedicated*, it is assumed that all resources (processors, memory, network, etc.) are used to process a single computational load [1,8–10,12,13,18, 21,22,51,55–59]. *Non-dedicated* systems may be used to simultaneously process different computational loads [9,10,52]. Most authors assume dedicated systems. In this case, an estimation of the available resources are used in the optimization process.

Some DLS variants assume a limitation to the maximum chunk size that can be received by each worker processor. When such a limitation does not exist, the DLS variant is said to be *unconstrained* [1,6,8,11–13,18,42,51–53,55, 56,60]. If the maximum chunk size is limited, the problem is said to be *buffer constrained* [9,10,19,21,22,41,43,57–59]. Many authors assume unconstrained systems, because there are usually plenty of secondary memory (e.g. hard disk drives) to temporarily store the load chunk at the worker. The MR-DLS variant studied in this work is unconstrained and assumes dedicated and heterogeneous worker processors. It was proved to be NP-hard in [43,60]. We present below the main heuristics in the literature of MR-DLS.

Yang et al. [59,61] were the first to tackle multi-round scheduling problems with heterogeneous workers. They proposed a heuristic approach called UMR. The workers are sorted in non-decreasing order of the value of  $G_i$ . This order is used as the transmission order. Let the *round length* of a worker i at a round jbe the sum of the setup times, the communication time, and the computation time of i at round j. UMR adds an additional constraint that the load chunk each active worker receives at a given round is such that the round length is the same for all active workers at that round. With this new constraint, the authors are able to derive closed-form formulas to compute the number of active workers n and the number of rounds m, as well as analytical expressions to compute the value of  $\theta_i^j$  for all  $i \in \{1, \ldots, n\}$  and  $j \in \{1, \ldots, m\}$ . We recall that  $\theta_i^j = 0$ , for all  $i \in \{n + 1, \ldots, |P|\}$ . Computational experiments showed that UMR outperforms the algorithms of [12] and [50], which were designed for homogeneous and single-round systems, respectively.

Hsu et al. [39] proposed a heuristic approach called Extended Smallest Communication Ratio (ESCR). The transmission order is computed as in [59, 61]. However, differently from [59, 61], all workers in P are activated, i.e., n = |P|. ESCR calculates the least common multiplier (LCM) of  $\{G_i + w_i | i \in P\}$  and uses this value to compute the round length. The number of rounds m is computed using closed-form formulas based on the LCM, and analytical expressions are used to compute the value of  $\theta_i^j$  for all  $i \in \{1, \ldots, n\}$  and  $j \in \{1, \ldots, m\}$ . Computational experiments showed that ESCR outperforms the algorithm of [7], which was designed for single-round systems.

Beaumont et al. [8] proposed and evaluated four different heuristics for MR-DLS. The one who obtained the best results was the Linear Programming with Adaptative Period heuristic (LPAP). The transmission order is computed as in [59,61]. However, the round length is not the same at each round. Instead, LPAP computes an upper bound T to the makespan. Based on this estimation, the length  $T_j$  of round j is set to  $T_j = \sqrt{T}$ . Once the round length is set, LPAP uses Linear Programming (LP) to find largest chunk size  $\theta_i^j$  that can be sent to each processor  $i \in P$  at the round j so that the round length is not larger than  $T_j$ . As the upper bound T is tighter at each iteration of this algorithm, the round length of a round j is always smaller than the length of the previous round. Computational experiments showed that LPAP outperforms the algorithm of [12].

Shokripour et al. [53] proposed two heuristics called Computational Based Method and Communication Based Method. Best results were obtained by the latter, referred here as CBM. Let the *round size* be the sum of the load chunks all the active workers receive at a given round. CBM adds an additional constraint that all rounds have the same size equal to Q, where Q is given by Equation (1).

$$Q = \frac{W}{m} \tag{1}$$

As before, the transmission order  $\pi$  has the workers sorted in non-decreasing order of  $G_i$ . The number of rounds m is given by the function  $q(n, \pi)$  shown in Equation (2).

$$q(n,\pi) = \sqrt{\frac{W\left[\left(1+\sum_{i=2}^{n} \Delta_{i}\right)\left(G_{1}+w_{1}\right)-\left(1+\sum_{i=2}^{n} E_{i}\right)\sum_{j=1}^{n} \Delta_{j}G_{j}\right]}{\left(1+\sum_{i=2}^{n} E_{i}\right)\left[\left(1+\sum_{i=2}^{n} \Delta_{i}\right)\left(\sum_{j=2}^{n} \phi_{j}G_{j}+\sum_{j=1}^{n} g_{j}\right)-\sum_{i=2}^{n} \phi_{i}\sum_{j=1}^{n} \Delta_{j}G_{j}\right]}}$$

$$(2)$$

The amount of load that can be processed using n workers and  $m = q(n, \pi)$  rounds, following the transmission order  $\pi$ , is estimated by the function  $f(n, \pi)$  shown in Equation (3). The number of processors is defined as the smallest value of n, such that  $f(n, \pi) \geq W$ .

$$f(n,\pi) = q(n,\pi) \cdot \left(\frac{(1+\sum_{i=2}^{n} \Delta_i)(\sum_{j=2}^{n} (\phi_j G_j + g_j) - s_1)}{w_1 - \sum_{j=2}^{n} \Delta_j G_j} + \sum_{i=2}^{n} \phi_i\right)$$
(3)

where,  $\Delta_i = (G_1 + w_1)/(G_i + w_i)$  and  $\phi_i = (G_1 + w_1)/(G_i + w_i)$ .

Once the values of n and m are computed. The percentage  $\alpha_i$  of the load size Q processed by each worker  $i \in \{1, \ldots, n\}$  at round  $j = \{1, \ldots, m-1\}$  is given by Equation (4), i.e.,  $\theta_i^j = Q \cdot \alpha_i$ .

$$\alpha_{i} = \begin{cases} (1 - \frac{1}{Q} \sum_{i=2}^{n} \phi_{i}) / (1 + \sum_{i=2}^{n} \Delta_{i}), & \text{if } i = 1, \\ \alpha_{1} \Delta_{i} + \frac{1}{Q} \phi_{i}, & \text{if } i = 2, ..., n. \end{cases}$$
(4)

Moreover, the percentage  $\beta_i$  of the load size Q processed by each worker  $i \in \{1, \ldots, n\}$  in the last round is given by Equation (5),  $\theta_i^m = Q \cdot \beta_i$ .

$$\beta_{i} = \begin{cases} (1 - \frac{1}{Q} \sum_{i=2}^{n} \Gamma_{i}) / (1 + \sum_{i=2}^{n} E_{i}), & \text{if } i = 1, \\ \beta_{1} E_{i} + \frac{1}{Q} \Gamma_{i}, & \text{if } i = 2, ..., n. \end{cases}$$
(5)

where,  $\delta_i = (s_i - s_{i+1} - g_{i+1})/(w_{i+1} + G_{i+1}), \epsilon_i = w_i/(w_{i+1} + G_{i+1}), E_i = \prod_{j=2}^i \epsilon_j,$ and  $\Gamma_i = \sum_{i=2}^i (\delta_j \prod_{k=j+1}^i \epsilon_k).$ The pseudo-code of CBM is presented in Figure 3. It receives as input the

The pseudo-code of CBM is presented in Figure 3. It receives as input the value W, the set P, and the values of  $g_i$ ,  $G_i$ ,  $s_i$ , and  $w_i$ , for all  $i \in P$ . The transmission order  $\pi$  is computed in line 1. The number n of active workers is computed in lines 2 to 5, while the number m of rounds is calculated in line 6. Besides, the round size Q is computed in line 7. The loop of lines 8 to 11 computes the value of  $\theta_i^j$  for all active worker  $i \in \{1, \ldots, n\}$ . The size of the load chunk received by i at the first m - 1 rounds is set in line 9, while that for the last round is set in line 10. The value of  $\theta_i^j$  for non-active workers is set in the loop of lines 12 to 14 The solution, represented by  $\pi$ , n, m and  $\theta_i^j$ , is returned in line 15. Computational experiments showed that this heuristic outperforms the heuristics LPAP [8] and ESCR [39]. As far as we can tell, CBM is the best heuristic in the literature of MR-DLS.

```
procedure CBM(W, P, g_i, G_i, s_i, w_i)
1
       Let \pi be a permutation of P with G_i \leq G_{i+1}, for all i \in \{1, \ldots, |P|-1\}
\mathbf{2}
      Set n \leftarrow 1
       while f(n, \pi) < W and n < |P| do
3
             n \leftarrow n+1
4
5
       end-while
6
      Set m \leftarrow q(n, \pi)
7
      Set Q \leftarrow W/m
8
       for i = 1, \ldots, n do
             Set \theta_i^j \leftarrow Q \cdot \alpha_i, for all j \in \{1, \ldots, m-1\}
9
             Set \theta_i^m \leftarrow Q \cdot \beta_i
10
11
      end-for
       for i = n + 1, ..., |P| do
12
             Set \theta_i^j \leftarrow 0, for all j \in \{1, \ldots, m\}
13
       end-for
14
      return < \pi, n, m, \theta_i^j >
15
end CBM
```

Fig. 3. Pseudo-code of the CBM heuristic for MR-DLS.

# 3 Biased random-key genetic algorithm

Genetic algorithms with random keys, or random-key genetic algorithms (RKGA), were first introduced by [4] for combinatorial optimization problems for which solutions can be represented as a permutation vector. 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 a RKGA. Parents are allowed to be selected for mating more than once in a given generation.

A biased random-key genetic algorithm (BRKGA) differs from a RKGA in the way parents are selected for crossover, see Gonçalves and Resende [29] 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 biased because one parent is always an elite solution and has a higher probability of passing its genes to the new generation.

The biased random-key genetic algorithm for MR-DLS, called GA-KEY, evolves a population of chromosomes that consists of vectors of real numbers (keys). Each solution is represented by a (|P| + 2)-vector, in which each key is a real number in the range [0, 1). Each of the first |P| keys are associated with a worker in P. The key |P| + 1 is associated with the number n of active workers, while the key |P| + 2 is associated with the number m of rounds. Each solution

represented by a chromosome is decoded by an algorithm that receives the vector of keys and builds a feasible solution for MR-DLS, i.e., the decoder returns a tuple  $\langle \pi, n, m, \theta_i^j \rangle$ , for all  $i \in \{1, \ldots, n\}$  and  $j \in \{1, \ldots, m\}$ .

Given a chromosome c, the decoding algorithm builds a solution as following. The transmission order  $\pi$  is obtained by sorting the workers in a non-decreasing order of their keys. Let x be the value of the (P+1)-th key in c, the number n of active workers is obtained uniformly from the value x, i.e.,  $n = \lfloor |P| \cdot x + 1 \rfloor$ . As there is no upper bound to the number of rounds, we use a different approach to compute the number m of rounds. Let y be the value of the (P+2)-th key in c, we have that  $m = \lfloor \frac{1}{1-y} \rfloor$ . An advantage of this approach is that smaller values of m are more likely to be chosen, which is opportune as the larger is the number of rounds, the larger is the amount of time spent in communication and computation setup. The values of  $\theta_i^j$ , for all  $i \in P$  and  $j \in \{1, \ldots, m\}$ , are calculated as in CBM [53], i.e., lines 6 to 14 of Figure 3. The makespan of the resulting solution is used as the fitness of the chromosome.

We use the parametric uniform crossover scheme proposed in [54] to combine two parent solutions and produce an offspring. In this scheme, the offspring inherits each of its keys from the best fit of the two parents with probability 0.6 and from the least fit parent with probability 0.4. This genetic algorithm 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: a fixed number of 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.

The (|P|+2) keys in the chromosome are randomly generated in the initial population. At each generation, the population is partitioned into two sets: TOP and REST. Consequently, 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 on the current population. As illustrated in Figure 4, 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 [4], 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 - NID|BOT offspring solutions are created. In our implementation, the population size was set to  $\gamma = |TOP| + |MID| + |BOT| = 5 \times |P|$ , with the sizes of sets TOP, *MID*, and *BOT* set to  $0.15 \times \gamma$ ,  $0.7 \times \gamma$ , and  $0.15 \times \gamma$ , respectively, as suggested by [14–17, 25–30, 32, 33, 37, 45, 49, 62].



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

# 4 Computational experiments

In this section, we report computational experiments to assess the performance of the biased random-key genetic algorithm GA-KEY. This algorithm was implemented in C++ and compiled with GNU C++ version 4.6.3. The experiments were performed on a 3.40GHz i7-4770 Intel Core CPU with 16GB of RAM memory.

Our experiments were carried out on the same set of instances used by [53]. The six instances have 50 workers and the value of W is 50, 250, 450, 650, 850, and 1050, respectively. The values  $g_i$ ,  $G_i$ ,  $s_i$ , and  $w_i$ ,  $i = 1, \ldots, |P|$ , were randomly generated, with  $G_i$  being twenty times greater than that of  $w_i$ .

In the first experiment, we investigate if GA-KEY efficiently identifies the relationships between keys and good solutions and converges to better solutions than those of CBM. For this end, we also compare its performance with that of a multi-start (MS-KEY) procedure that uses the same decoding heuristic as GA-KEY. Each iteration of MS-KEY applies this same decoding heuristic starting from a randomly generated vector of random-keys. Therefore, nothing is learned from one iteration to the next. GA-KEY and MS-KEY were run 10 times for each instance, with different seeds for the random number generator [44]. The former was made to stop after |P| generations without improvement in the cost of the best solution found, while the latter stopped after  $|P| \times \gamma$  iterations without improvement, where  $\gamma$  is the population size of GA-KEY.

The results of this experiment is shown in Table 1. The first column identifies the instance. The next two columns provide the makespan and CPU time in seconds obtained by CBM. The average makespan, relative improvement ([CBM-MS-KEY]/CBM), and CPU time in seconds obtained by MS-KEY are displayed in columns 4 to 6. The same data is displayed for GA-KEY in the last three columns. It can be seen that GA-KEY obtained the best results for all instances. The average relative improvement of GA-KEY over CBM was up to 18.85%. The average relative improvement observed for MS-KEY and GA-KEY were, respectively, 7.74% and 11.68%. Besides, the maximum CPU time observed by GA-KEY was 0.22 seconds. Moreover, Figure 5 shows six plots of CPU time versus best solution known of CBM, MS-KEY, and GA-KEY on the six instances studied. They show that GA-KEY systematically finds better solutions faster than the other heuristics, and that it converges to its best solution known before one second on all instances.

Table 1. Results of GA-KEY and MS-KEY compared to those of CBM.

	CBM			MS-KEY		GA-KEY		
id	makespan	time(s)	makespan	gap(%)	time(s)	makespan	gap(%)	time(s)
W-50	1367.50	0.01	1183.85	13.43	0.25	1109.77	18.85	0.20
W-250	4614.69	0.01	4124.45	10.62	0.24	3954.85	14.30	0.21
W-450	7537.98	0.01	6865.69	8.92	0.24	6599.60	12.45	0.21
W-650	10203.79	0.01	9600.56	5.91	0.28	9228.82	9.55	0.21
W-850	12887.14	0.01	12333.86	4.29	0.25	11861.24	7.96	0.21
W-1050	15578.63	0.01	15065.73	3.29	0.25	14492.70	6.97	0.22
Average:	8698.29	0.01	8195.69	7.74	0.25	7874.50	11.68	0.21

In the second experiment, we evaluate the run time distributions or timeto-target plots (TTT plots, for short) of MS-KEY and GA-KEY. TTT 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 [38] as a way to characterize the running times of stochastic local search algorithms for combinatorial optimization. In this experiment, both MS-KEY and GA-KEY were made to stop whenever a solution with cost smaller than or equal to a given target value was found. The target is set as the cost of the best known solution for the instance. The heuristics were run 200 times for each instance, with different 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 described in [2,3]. We associate a probability  $p_i = (i - \frac{1}{2})/200$  with the *i*-th smallest running time  $t_i$  and plot the points  $G_i = (t_i, p_i)$ , for i = 1, ..., 200. The more to the left is a plot, the better is the algorithm corresponding to it.

The TTT plots of the six instances studied are shown in Figure 6. It can be seen that GA-KEY finds solutions with makespan for W-50 equal to 1109.28, 3954.92 to W-250, and 6595.53 to W-450, all with a probability close to 100% in less than one second of CPU time, while MS-KEY may take up to 60 seconds to find solutions as good as those with the same probability. Similar results can



Fig. 5. CPU time versus best solution known for the first six seconds of CBM, MS-KEY, and GA-KEY on instances (a) W-50, (b) W-250, (c) W-450, (d) W-650, (e) W-850, and (f) W-1050.

be observed for the other instances. GA-KEY finds solutions with makespan for W-650 equal to 9230.49, 11860.48 to W-850 and 14492.37 to W-1050, all with a probability close to 100% in less than five seconds of CPU time, while MS-KEY may take up to 120.0 seconds to find solutions as good as them with the same probability. Moreover, we also used the tool proposed by Ribeiro et al. [47] to perform a direct numerical comparison of MS-KEY and GA-KEY. Let  $T_{GA-KEY}$  (resp.  $T_{MS-KEY}$ ) be the continuous random variable representing the time needed by GA-KEY (resp. MS-KEY) to find a solution as good as the target, and let  $P(T_{GA-KEY} \leq T_{MS-KEY})$  be the probability that GA-KEY converges faster than MS-KEY. This probability is computed by the software available in [48] and reported in the caption of Figures 6 for six instances studied. On can see from this result that the probability of GA-KEY be faster then MS-KEY is at least 96%. These results show that BRKGA identifies the best relationships between the keys and the good solutions throughout the evolutionary process, converging to solutions better then the other heuristics studied.

# 5 Concluding Remarks

We considered the unconstrained multi-round divisible load scheduling problem with dedicated and heterogeneous processors. In this case, the transmission is divided into rounds and every active worker receives a load chunk at each round. Multi-round systems may reduce the makespan by decreasing the idle times, at the additional cost of repeated setup times at each round. A new heuristic biased random-key genetic algorithm has been proposed for the problem: GA-KEY. The GA-KEY heuristic improves upon the best algorithm in the literature, CBM, and MS-KEY multistart heuristic in terms of solution quality, since it performs a global search in the space of processor permutations in order to find the best transmission order of the processors and the values of the n and m. Computational experiments showed that the proposed heuristic outperforms the CBM by 11.68%, on average.

### References

- 1. 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, pp. 54–61. Nashville (2009)
- Aiex, R., Resende, M., Ribeiro, C.C.: Probability distribution of solution time in GRASP: An experimental investigation. Journal of Heuristics 8, 343–373 (2002)
- Aiex, R., Resende, M., Ribeiro, C.C.: TTTPLOTS: A Perl program to create timeto-target plots. Optimization Letters 1, 355–366 (2007)
- Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 2, 154–160 (1994)
- Beaumont, O., Bonichon, N., Eyraud-Dubois, L.: Scheduling divisible workloads on heterogeneous platforms under bounded multi-port model. In: International Symposium on Parallel and Distributed Processing, pp. 1–7. IEEE, Miami (2008)



Fig. 6. Time-to-target plots for instances (a) W-50, with a target of 1109.28 and  $P(T_{GA-KEY} \leq T_{MS-KEY}) = 0.960$ , (b) W-250, with a target of 3954.92 and  $P(T_{GA-KEY} \leq T_{MS-KEY}) = 0.967$ , (c) W-450, with a target of 6595.53 and  $P(T_{GA-KEY} \leq T_{MS-KEY}) = 0.980$ , (d) W-650, with a target of 9230.49 and  $P(T_{GA-KEY} \leq T_{MS-KEY}) = 0.990$ , (e) W-850, with a target of 11860.48 and  $P(T_{GA-KEY} \leq T_{MS-KEY}) = 0.989$ , and (f) W-1050, with a target of 14492.37 and  $P(T_{GA-KEY} \leq T_{MS-KEY}) = 0.988$ .
- Beaumont, O., Casanova, H., Legrand, A., Robert, Y., Yang, Y.: Scheduling divisible loads on star and tree networks: results and open problems. IEEE Transactions on Parallel and Distributed Systems 16, 207–218 (2005)
- Beaumont, O., Legrand, A., Robert, Y.: The master-slave paradigm with heterogeneous processors. IEEE Transactions on Parallel and Distributed Systems 14(9), 897–908 (2003)
- Beaumont, O., Legrand, A., Robert, Y.: Optimal algorithms for scheduling divisible workloads on heterogeneous systems. In: 12th Heterogeneous Computing Workshop, pp. 98–111. IEEE Computer Society Press, Nice (2003)
- Berlinska, J., Drozdowski, M.: Heuristics for multi-round divisible loads scheduling with limited memory. Parallel Computing 36, 199–211 (2010)
- Berlińska, J., Drozdowski, M., Lawenda, M.: Experimental study of scheduling with memory constraints using hybrid methods. Journal of Computational and Applied Mathematics 232, 638–654 (2009)
- Bharadwaj, V., Ghose, D., Mani, V.: Multi-installment load distribution in tree networks with delays. IEEE Transactions on Aerospace and Electronic Systems 31, 555–567 (1995)
- Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.G.: Scheduling divisible loads in parallel and distributed systems. Wiley - IEEE Computer Society Press (1996)
- Blazewicz, J., Drozdowski, M.: Distributed processing of divisible jobs with communication startup costs. Discrete Applied Mathematics 76, 21–41 (1997)
- Brandão, J.S., Noronha, T.F., Resende, M.G.C., Ribeiro, C.C.: A biased randomkey genetic algorithm for single-round divisible load scheduling. International Transactions Operational Research 22, 823–839 (2015)
- Buriol, L.S., Resende, M.G., Ribeiro, C.C., Thorup, M.: A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. Networks 46, 36–56 (2005)
- Buriol, L.S., Resende, M.G., Thorup, M.: Survivable IP network design with OSPF routing. Networks 49, 51–64 (2007)
- Chan, F., Tibrewal, R.K., Prakash, A., Tiwari, M.: A biased random key genetic algorithm approach for inventory-based multi-item lot-sizing problem. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 229(1), 157–171 (2015)
- Drozdowski, M.: Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems. 321. Politechnika Poznanska (1997)
- Drozdowski, M., Lawenda, M.: Multi-installment divisible load processing in heterogeneous systems with limited memory. Parallel Processing and Applied Mathematics **3911**, 847–854 (2006)
- Drozdowski, M., Lawenda, M.: Multi-installment divisible load processing in heterogeneous distributed systems. Concurrency and Computation: Practice and Experience 19(17), 2237–2253 (2007)
- Drozdowski, M., Wolniewicz, P.: Divisible load scheduling in systems with limited memory. Cluster Computing 6, 19–29 (2003)
- Drozdowski, M., Wolniewicz, P.: Optimum divisible load scheduling on heterogeneous stars with limited memory. European Journal of Operational Research 172, 545–559 (2006)
- Duarte, A., Mart, R., Resende, M.G.C., Silva, R.M.A.: Improved heuristics for the regenerator location problem. International Transactions in Operational Research 21, 541–558 (2014)
- 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, 299– 333 (2002)

- Festa, P., Gonçalves, J.F., Resende, M.G.C., Silva, R.M.A.: Automatic tuning of grasp with path-relinking heuristics with a biased random-key genetic algorithm. In: Experimental Algorithms, pp. 338–349. Springer (2010)
- Fontes, D.B., Gonçalves, J.F.: A multi-population genetic algorithm for tree-shaped network design problems. In: In Proceedings: IJCCI 2009 - International Joint Conference on Computational Intelligence, pp. 177–182 (2009)
- Gonçalves, J.F., de Magalhães Mendes, J.J., Resende, M.G.C.: A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research 167, 77–95 (2005)
- Gonçalves, J.F., Mendes, J.d.M., Resende, M.G.C.: A genetic algorithm for the resource constrained multi-project scheduling problem. European Journal of Operational Research 189(3), 1171–1190 (2009)
- Gonçalves, J.F., Resende, M.G.C.: Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics 17, 487–525 (2011)
- Gonçalves, J.F., Resende, M.G.C.: A biased random key genetic algorithm for 2D and 3D bin packing problems. International Journal of Production Economics 145, 500–510 (2013)
- Gonçalves, J.F., Resende, M.G.C.: An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. International Transactions in Operational Research 21, 215–246 (2014)
- Gonçalves, J.F., Resende, M.G.C., Costa, M.D.: A biased random-key genetic algorithm for the minimization of open stacks problem. International Transactions in Operational Research (2014)
- 33. Gonçalves, J.F., Resende, M.G.C., Mendes, J.J.: A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. Journal of Heuristics 17(5), 467–486 (2011)
- Gonçalves, J.F., Resende, M.G.C., Toso, R.F.: Biased and unbiased random-key genetic algorithms: An experimental analysis. Tech. rep., AT&T Labs Research, Florham Park (2012)
- Gonçalves, J.F., Resende, M.G.C.: An evolutionary algorithm for manufacturing cell formation. Computers and Industrial Engineering 47, 247–273 (2004)
- Gonçalves, J.F., Resende, M.G.C., Costa, M.D.: A biased random-key genetic algorithm for the minimization of open stacks problem. International Transactions in Operational Research (2014). DOI 10.1111/itor.12109
- 37. Goulart, N., de Souza, S.R., Dias, L.G.S., Noronha, T.F.: Biased random-key genetic algorithm for fiber installation in optical network optimization. In: Proceedings of the 2011 IEEE Congress on Evolutionary Computation, pp. 2267–2271. New Orleans (2011)
- Hoos, H., Stützle, T.: Evaluation of Las Vegas algorithms Pitfalls and remedies. In: G. Cooper, S. Moral (eds.) Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, pp. 238–245. Madison (1998)
- Hsu, C.H., Chen, T.L., Park, J.H.: On improving resource utilization and system throughput of master slave job scheduling in heterogeneous systems. The Journal of Supercomputing 45(1), 129–150 (2008)
- Kim, H.J.: A novel optimal load distribution algorithm for divisible loads. Cluster Computing- The Journal of Networks Software Tools and Applications 6, 41–46 (2003)
- Li, X., Bharadwaj, V., Ko, C.: Divisible load scheduling on single-level tree networks with buffer constraints. IEEE Transactions on Aerospace and Electronic Systems 36, 1298–1308 (2000)

- Lin, X., Deogun, J., Lu, Y., Goddard, S.: Multi-round real-time divisible load scheduling for clusters. In: High Performance Computing-HiPC 2008, pp. 196– 207. Springer (2008)
- Maciej, D., Marcin, L.: Scheduling multiple divisible loads in homogeneous star systems. Journal of Scheduling 11, 347–356 (2008)
- Matsumoto, M., Nishimura, T.: Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation 8, 3–30 (1998)
- Noronha, T.F., Resende, M.G.C., Ribeiro, C.C.: A biased random-key genetic algorithm for routing and wavelength assignment. Journal of Global Optimization 50, 503–518 (2011)
- Resende, M.G.C.: Biased random-key genetic algorithms with applications in telecommunications. TOP 20, 130–153 (2012)
- Ribeiro, C.C., Rosseti, I., Vallejos, R.: Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. Journal of Global Optimization 54, 405–429 (2012)
- Ribeiro, C.C., Rosseti, I., Vallejos, R.: tttplots-compare: A perl program to compare time-to-target plots or general runtime distributions of randomized algorithms (2014). Online reference at http://www2.ic.uff.br/~celso/tttplots/index.html, last visited on January 25, 2014.
- Roque, L., Fontes, D., Fontes, F.: A hybrid biased random key genetic algorithm approach for the unit commitment problem. Journal of Combinatorial Optimization 28(1), 140–166 (2014)
- 50. Rosenberg, A.L.: Sharing partitionable workloads in heterogeneous nows: greedier is not better. In: cluster, p. 124. IEEE (2001)
- Shokripour, A., Othman, M., Ibrahim, H.: A method for scheduling last installment in a heterogeneous multi-installment system. In: Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology, pp. 714–718. Chengdu (2010)
- Shokripour, A., Othman, M., Ibrahim, H., Subramaniam, S.: A new method for job scheduling in a non-dedicated heterogeneous system. Procedia Computer Science 3, 271–275 (2011)
- Shokripour, A., Othman, M., Ibrahim, H., Subramaniam, S.: New method for scheduling heterogeneous multi-installment systems. Future Generation Computer Systems 28, 1205–1216 (2012)
- 54. Spears, W., deJong, K.: 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. Morgan Kaufman, San Mateo (1991)
- 55. Wang, M., Wang, X., Meng, K., Wang, Y.: New model and genetic algorithm for divisible load scheduling in heterogeneous distributed systems. International Journal of Pattern Recognition and Artificial Intelligence 27 (2013)
- Wang, X., Wang, Y., Meng, K.: Optimization algorithm for divisible load scheduling on heterogeneous star networks. Journal of Software 9, 1757–1766 (2014)
- 57. Wolniewicz, P.: Divisible job scheduling in systems with limited memory. Ph.D. thesis, Poznan University of Technology, Poznań (2003)
- Yang, Y., Casanova, H.: RUMR: Robust scheduling for divisible workloads. In: Proceedings of the 12th IEEE Symposium on High Performance and Distributed Computing, pp. 114–125. Seattle (2003)
- Yang, Y., Casanova, H.: UMR: A multi-round algorithm for scheduling divisible workloads. In: Proceedings of the 17th International Parallel and Distributed Processing Symposium, pp. 24–32. Nice (2003)

- 60. Yang, Y., Casanova, H., Drozdowski, M., Lawenda, M., Legrand, A., et al.: On the complexity of multi-round divisible load scheduling. Tech. Rep. 6096, INRIA Rhône-Alpes (2007)
- Yang, Y., van der Raadt, K., CasaNova, H.: Multi-round algorithms scheduling divisible loads. IEEE Transactions on Parallel and Distributed Systems 16, 1092– 1102 (2005)
- Zheng, J.N., Chien, C.F., Gen, M.: Multi-objective multi-population biased random-key genetic algorithm for the 3-d container loading problem. Computers & Industrial Engineering (2014)