Pesquisa Operacional

Prof. Rian Gabriel S. Pinheiro, 2020.2. *http://professor.ufal.br/rian/*

Algoritmo Genético BRKGA para o Problema de Empacotamento 2D Clássico com desempate

Rian G.S. Pinheiro
rian@ic.ufal.br

Eric Coelho
esc2@ic.ufal.br

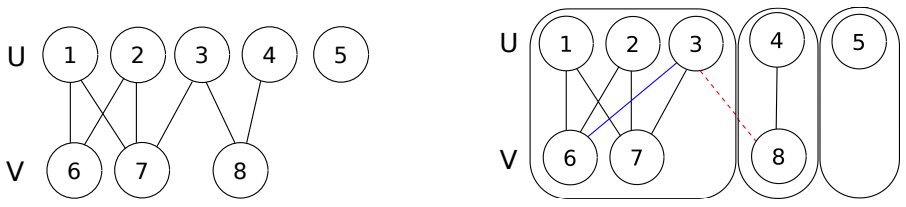
Resumo

O *Problema de empacotamento* (BPP) consiste em armazenar ortogonalmente um conjunto de itens na menor quantidade de caixas possível. A versão clássica assume que os itens têm orientação fixa e não pode haver sobreposição no empacotamento. O caso bidimensional generaliza o problema de empacotamento unidimensional clássico amplamente difundido na literatura, portanto, é da classe $\mathcal{NP}$-Difícil . Este trabalho apresenta um algoritmo genético de chaves aleatórias viciadas (BRKGA) para o *problema de empacotamento bidimensional* (2BP), considerando os itens alocados com orientação fixa, baseando-se em chaves aleatórias como critério de desempate entre os espaços vazios ao inserir um novo item. O problema tem várias aplicações industriais, como corte, repartição e agendamento, relacionando-se a outros problemas complexos.

Palavras-chave: BRKGA, Bin Packing, 2BP, Meta-heurísticas.

1 Introdução

O *problema de empacotamento clássico bidimensional* (2BP) consiste em empacotar ortogonalmente um conjunto de n itens em forma de retângulos caracterizados por sua altura h_i e largura w_i, $i \in \{1, 2, 3, ..., n\}$, no menor número possível de caixas homogêneas de altura $H \geq hi$ e largura $W \geq wi$. A quantidade de caixas é ilimitada e os itens não podem ser alocados com sobreposição.



(a) Instância. (b) Solução.

Figura 1: Exemplo do BGEP.

A Figura 1 mostra um exemplo de um grafo que com uma adição, aresta $(3, 6)$

1.1 Aplicações

empty

2 Algoritmo BRKGA

O algoritmo genético de chaves aleatórias viciadas (BRKGA) foi introduzido por Gonçalves & Resende (2013) para problemas de otimização combinatória. Esta meta-heurística é uma variante do algoritmo genético de chaves aleatórias (RKGA) [Bean, 1994]. A ideia principal é evoluir uma população codificada. Os indivíduos são representados por cromossomos, denominado chaves aleatórias ou random-keys, na forma de vetores de números reais no intervalo [0, 1]. A implementação desse processo requer um componente chamado decodificador. Este componente deve receber um vetor de random-keys para fornecer uma representação do indivíduo que pode ser avaliada. O decodificador é o único componente que depende das características do problema. Assim, duas implementações da meta-heurística para diferentes problemas só necessitam de decodificadores diferentes. Para o 2BP, o decodificador é composto de dois componentes: um método de ordenação e um algoritmo guloso para realizar o empacotamento dos itens. Tal algoritmo guloso constrói uma solução iterativamente, recebendo uma permutação de inteiros que representa a sequência em que os itens serão empacotados sequencialmente. O método de ordenação deve receber um vetor com n random-keys, um para cada item da instância, para fornecer uma permutação de itens. As chaves devem ser ordenadas de modo que o item ai é empacotado antes do item aj se e somente se $k_i < k_j, i, j \in \{1, ..., n\}$, onde k_i e k_j são as respectivas random-keys associadas.

Após o processo de ordenação, a permutação associada ao cromossomo do indivíduo é passada como parâmetro de entrada para o algoritmo guloso, denominado Distance to the Front-Top-Right Corner (DFTRC). Sabendo que uma caixa aberta da solução corrente é uma que contém itens empacotados, para cada iteração k do DFTRC, procuramos em cada caixa aberta por áreas que podem conter o respectivo item $a_k, k \in \{1, ..., n\}$. Se nenhuma caixa aberta até o momento pode conter o item, então uma nova caixa é introduzida na solução com a_k empacotado. No passo de empacotamento, áreas maximais são utilizadas para modelar o espaço vazio de cada caixa aberta. Uma área maximal é a maior área retangular possível dentro da caixa aberta que não há itens. Todas as áreas maximais que podem conter o item da iteração corrente são candidatas para a escolha da localização do empacotamento. O DFTRC seleciona a área maximal que maximiza a distância entre o canto superior direito frontal do item para o respectivo canto superior direito frontal da caixa.

Caso haja empate entre as áreas maximais, um segundo vetor de random keys é utilizado para selecionar entre as duas melhores áreas maximais.

O Algoritmo 3 apresenta a visão geral da heurística.

2.1 Método Construtivo

Dado como entrada a árvore geradora mínima T, as soluções iniciais de cada iteração do GRASP são construídas com base em remoções aleatórias das arestas desta árvore. Na floresta geradora resultante, cada árvore definirá um agrupamento U_i para compor a biclique.

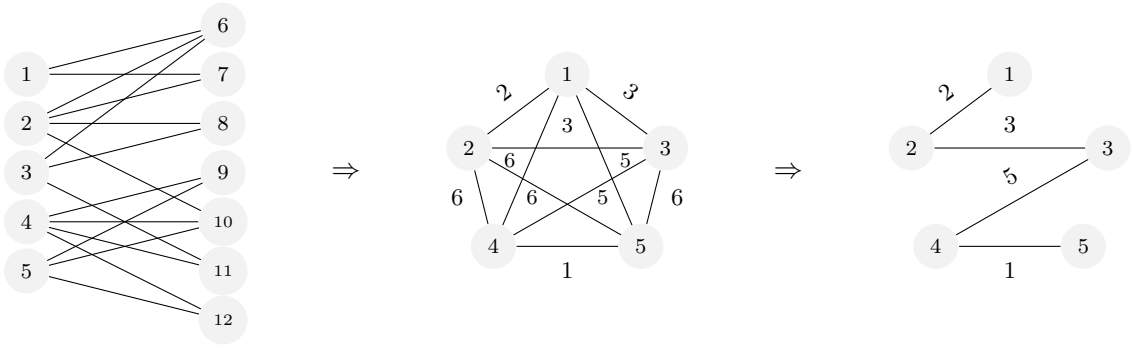


Figura 2: Execução do Algoritmo Construtivo

Com base nisso, para cada vértice em V, verifica-se em qual agrupamento U_i deverá fazer parte, de maneira a conter o menor número de edições. Isto é feito associando o vértice em questão a cada uma dos conjuntos de U_i e verificando o quanto de remoções e adições de arestas são necessárias para que esse vértice se ligue com todos os vértices do conjunto associado.

O agrupamento U_i em que seja necessário o menor número de edições será o ideal e portanto escolhido. Em seguida, combinamos U e V para formarmos a biclusterização $\{U, V\}$ ligando as arestas no grafo G. No algoritmo 1 apresentamos o método construtivo.

Algoritmo 1 Método Construtivo

procedure CONSTRUTIVO(T, k)
  while $E(T) \neq \emptyset$ **do**
    $\mu \leftarrow \{\infty, \cdots, \infty\}_{1\times|E(T)|}$
    for all $i \in E(T)$ **do**
      $T_i \leftarrow T - \{i\}$
      construa U_i a partir de T_i ▷ um biclique para cada componente conexa de T_i
      construa V_i ideal a partir de U_i ▷ V_i que tem menor número de erros dado U_i
      $C_i \leftarrow$ BICLUSTER(U_i, V_i) ▷ combina os agrupamentos U_i e V_i para gerar a biclique C_i
      $\mu_i \leftarrow$ NUMEDICOES(C_i)
    end for
    escolha $\tilde{C}$ aleatoriamente entre os k melhores C_i ▷ baseado nos valores em μ
    atualiza T com base no $\tilde{C}$ escolhido
    if $\tilde{\mu} < \mu^*$ **then**
      $C^* \leftarrow \tilde{C}$
    end if
  end while
  return C^*
end procedure

2.2 Busca Local

Após o método construtivo, inicia-se o método de Busca Local baseado no VND (Lodi et al., 2002). Tendo como entrada um solução inicial, a Busca Local tenta melhorar a solução através da exploração do espaço de solução. Dado um conjunto de estruturas de vizinhança, o método de Busca Local explora o espaço de solução trocando sistematicamente as estruturas de vizinhança.

O método aceita somente soluções que melhorem a solução corrente. Nestes casos, o método volta à primeira estrutura de vizinhança e inicia uma nova busca. Usamos aqui uma variante do VND que consiste em usar uma ordenação aleatória dessas estruturas, como mostrado no Algoritmo 2.

O método proposto utiliza três estruturas de vizinhança: "Reagrupe"; "Mude de Biclique" e "Troca de Vértice". A complexidade computacional da busca em cada uma dessas vizinhanças é de $O(n^2)$. Elas são presentadas a seguir.

Reagrupe: Dado os conjuntos de grupos de vértices, formados pela intersecção de cada uma das partes, U por exemplo, com cada biclique B da solução atual. A Busca Local Reagrupe aloca os vértices da outra parte, no caso V, ao grupo de vértice de forma a minimizar o número de edições.

Muda de Biclique: Esta Busca Local move um vértice de uma biclique para outra.

Troca de Vértice: Esta Busca Local troca dois vértices (de mesma parte) de uma biclique para a outra. Ao contrário da "Mude de Biclique" não possibilita a eliminação de bicliques previamente existentes.

Algoritmo 2 VND com ordem aleatórias

```
1: procedure VND(A)
2:     inicialize V randomicamente
3:     k ← 1
4:     μ* ← μ(A)
5:     repeat
6:         A' ← V^k(A)
7:         if μ(A') > μ* then
8:             A* ← A'
9:             k ← 1
10:        else
11:            k ← k + 1
12:        end if
13:    until k = k_max
14:    return A*
15: end procedure
```

3 Experimentos Computacionais

A base de testes utilizada neste trabalho é composta por 500 instâncias do 2BP detalhadas em Martello & Vigo (1998). As instâncias são organizadas em 10 classes com 10 instâncias para cada quantidade de itens $n \in \{20, 40, 60, 80, 100\}$. Esta base se trata de uma extensão das instâncias propostas em Berkey & Wang (1987).

3.1 Ferramentas

O BRKGA proposto foi implementado utilizando a linguagem de programação C++11, e foi compilado com o GCC do GNU Compiler Collection. O ambiente computacional utilizado em todos os testes neste trabalho consiste de um desktop munido da seguinte configuração: processador AMD FX-6300 @3.5 GHz, 8 GB de memória RAM e sistema operacional Ubuntu 18.04. A Tabela 1 relaciona os parâmetros calibráveis do BRKGA com os respectivos valores.

Algoritmo 3 Visão Geral da Heurística GRASP

```
1: procedure MST+BL(G, k)
2:     it ← 0
3:     μ* ← ∞
4:     transforme o grafo G numa clique C com pesos d_ij
5:     calcule a árvore geradora de custo mínimo T de C
6:     repeat
7:         C_it ← Construtivo(T, k)          ▷ k = 1, guloso; k = m, aleatório; 1 < k < m, k-melhores
8:         C'_it ← BuscaLocal(C_it)
9:         μ_it ← NumEdicoes(C'_it)          ▷ número de edições (remoção ou adição de arestas)
10:        if μ_it < μ* then
11:            C* ← C'_it
12:        end if
13:        it ← it + 1
14:    until it = 100
15:    return C*
16: end procedure
```

3.2 Resultado

TABELA de RESULTADOS DA EXECUÇÃO

A Tabela 1 apresenta a instância, seguida do número de edições encontradas pelo algoritmo exato Lodi et al. (2002), a solução encontrada na literatura (Lai & Chan, 1997), juntamente com a solução do GRASP proposto e do GRASP+SP. Observa-se que o algoritmo GRASP encontra o valor ótimo em todas as instâncias com exceção da `i11`. Já o GRASP+SP, encontrou a solução ótima em todas as instâncias.

A Tabela 2 apresenta a instância, junto com o tempo de execução do algoritmo de Lodi et al. (2002), o tempo de execução do GRASP proposto e o tempo de execução do GRASP+SP. Note que, o tempo de GRASP proposto foi muito inferior ao tempo de execução da literatura. Observa-se ainda que os tempos do GRASP e do GRASP+SP foram próximos. Além disso, algoritmo de foi executado em uma máquina Intel Core i7-2600 com 3.40GHz, enquanto que o deste artigo foi executado em uma máquina Intel Core i7-4500U com 1.80GHz.

No que diz respeito ao pré-processamento da Seção 2, ele se mostrou eficaz em instâncias esparsas, porém sem muita efetividade para as instâncias densas.

4 Conclusão

Este trabalho aplicou a heurística BRKGA para o problema clássico de empacotamento bidimensional, sem considerar a rotação dos itens. O algoritmo é baseado na meta-heurística BRKGA, e utiliza um algoritmo guloso para empacotar os itens. Através de um conjunto de 500 instâncias, as quais compõem a base de teste padrão para vários trabalhos encontrados na literatura, o BRKGA foi comparado com outras abordagens encontradas na literatura. Os experimentos mostraram que o BRKGA proposto aliado com o critério de desempate obtém resultados equivalentes ou melhores aos reportados na literatura.

PROPROSTAS Para trabalhos futuros serão propostas novas melhorias que acelerem o método, como por exemplo novo módulos exatos. Pretende-se estender o pré-processamento proposto para os coasos em que a biclique não é completa. Além disso, um estudo mais aprofundado com relação à elaboração de instâncias mais difíceis. Com relação a meta-heurística, pode-se pensar em novos métodos construtivos e novas buscas locais.

Tabela 1: Resultados do GRASP e GRASP+SP

Instância	Exato	(Lai & Chan, 1997)	GRASP	GRASP+SP
i1	17	17	17	17
i2	20	20	20	20
i3	36	36	36	36
i4	34	34	34	34
i5	35	36	35	35
i6	30	30	30	30
i7	50	51	50	50
i8	28	28	28	28
i9	34	34	34	34
i10	52	53	52	52
i11	53	54	54	**53**
i12	61	62	61	61
i13	54	54	54	54
i14	63	64	63	63
i15	61	61	61	61
i16	26	28	26	26
i17	40	40	40	40
i18	34	35	34	34
i19	49	49	49	49
i20	51	52	51	51
i21	88	88	88	88
i22	74	74	74	74
i23	24	24	24	24
i24	24	24	24	24
i25	94	94	94	94
i26	23	23	23	23
i27	111	111	111	111
i28	62	63	62	62
i29	56	56	56	56
i30	97	97	97	97

Referências

Berkey, J. O. & Wang, P. Y. (1987), 'Two-dimensional finite bin-packing algorithms', *Journal of the operational research society.*

Gonçalves, J. F. & Resende, M. G. (2013), 'A biased random key genetic algorithm for 2d and 3d bin packing problems', *International Journal of Production Economics.*

Lai, K. & Chan, J. (1997), 'Developing a simulated annealing algorithm for the cutting stock problem', *Computers and Industrial Engineering.*

Lodi, A., Martello, S. & Monaci, M. (2002), 'Two-dimensional packing problems: A survey', *European journal of operational research.*

Martello, S. & Vigo, D. (1998), 'Exact solution of the two-dimensional finite bin packing problem', *Management science.*

Zudio, A., Costa, D., Masquio, B., Coelho, I. & Pinto, P. (2018), 'Brkga/vnd hybrid algorithm for the classic three-dimensional bin packing problem', *Electronic Notes in Discrete Mathematics.*

Tabela 2: Tempos do GRASP e GRASP+SP

Instância	(Zudio et al., 2018)	GRASP	GRASP+SP
i1	7,23	0,26	0,30
i2	7,00	0,21	0,24
i3	10,09	0,39	0,45
i4	9,97	0,35	0,41
i5	12,90	0,41	0,46
i6	13,89	0,48	0,49
i7	7,88	0,49	0,52
i8	13,13	0,39	0,42
i9	12,53	0,47	0,50
i10	20,31	0,98	1,09
i11	23,41	1,35	1,17
i12	24,50	1,40	1,26
i13	47,19	2,66	2,39
i14	48,24	2,82	2,78
i15	65,95	3,27	3,25
i16	3,40	0,09	0,10
i17	4,02	0,05	0,05
i18	5,26	0,12	0,14
i19	5,28	0,06	0,08
i20	7,05	0,36	0,39
i21	31,80	0,15	0,14
i22	24,27	0,17	0,14
i23	26,45	1,71	1,48
i24	25,56	3,28	3,24
i25	26,48	0,16	0,19
i26	43,42	2,95	3,05
i27	51,80	0,20	0,22
i28	77,30	15,52	16,09
i29	71,91	13,73	13,65
i30	77,21	0,34	0,37

Pesquisa Operacional
Instituto de Computação
Universidade Federal de Alagoas

Histórico do Artigo:
Versão 1
Entregue: 18 de fevereiro de 2020