

## Trabalho Final de Otimização Combinatória (2019/1)

### 1 Definição

O trabalho final consiste no estudo e implementação de uma meta-heurística sobre um problema  $\mathcal{NP}$ -completo, e na formulação matemática em programação linear inteira mista deste problema. O trabalho deve ser realizado em grupos, e cada grupo escolhe uma combinação (*problema, meta-heurística*). Não há restrições quanto a repetição de problemas e técnicas, mas cada grupo deve ter um combinação única de problema e técnica de resolução.

O trabalho é dividido em quatro partes: (i) formulação matemática, (ii) implementação da meta-heurística, (iii) relatório, e (iv) apresentação em aula das partes (i) e (ii) e dos resultados obtidos. Informações sobre cada parte são detalhadas nas seções a seguir. A definição dos problemas e instâncias, bem como alguns materiais adicionais, também se encontram neste documento.

A entrega deverá ser feita pelo *moodle* da disciplina, até a data previamente definida. As quatro partes devem ser entregues em um arquivo compactado (formatos `.tar.gz`, `.zip`, `.rar`, ou `.7z`). Note que para a implementação, deve ser entregue o código fonte e **não** o executável.

### 2 Formulação Matemática

O problema escolhido pelo grupo deve ser formulado matematicamente em Programação Linear Inteira Mista, e esta deve ser implementada em GNU MathProg para ser executada no GLPK. O arquivo de formulação **deve** ser separado do arquivo de dados, como visto em aula de laboratório. A entrega deve conter um arquivo de modelo (arquivo `.mod`), e os arquivos de dados das instâncias, já convertidos para o padrão do GLPK e de acordo com a formulação proposta.

### 3 Implementação

A implementação do algoritmo proposto pode ser feita nas **linguagens de programação gratuitas** (C, C++, Java, Python, Julia, etc.), **sem o uso de bibliotecas proprietárias**. A compilação e execução dos códigos deve ser possível em ambiente Linux **ou** Windows, pelo menos. Além disso, alguns critérios básicos devem ser levados em conta no desenvolvimento:

- Critérios de engenharia de software: documentação e legibilidade, mas sem exageros;
- Os parâmetros do método de resolução devem ser **recebidos via linha de comando**<sup>1</sup>, em especial a semente para o gerador de números pseudo aleatórios;

<sup>1</sup>Na linguagem C++, por exemplo, os parâmetros da linha de comando são recebidos com uso de `argc` e `argv` na função `main`.

- Critérios como qualidade das soluções encontradas e eficiência da implementação serão levados em conta na avaliação (i.e., quando modificar uma solução, calcular a diferença de custo do vizinho, e não o custo de toda a solução novamente).
- O critério de parada do algoritmo **não** deve ser tempo de execução. Alguns dos critérios de parada permitidos são: número de iterações, número de iterações sem encontrar uma solução melhor, ou proximidade com algum limitante, ou ainda a combinação de algum dos anteriores. Estes critérios devem ser calibrados de forma a evitar que o algoritmo demore mais do que **cinco minutos** para executar nas instâncias fornecidas;

A entrega da implementação é o **código fonte**. Junto com ele deve haver um arquivo README informando como compilar/executar o código, e se são necessárias quaisquer bibliotecas específicas (Boost, Apache Commons, etc.).

## 4 Relatório

O relatório, a ser entregue em formato PDF, deve possuir no máximo seis páginas (sem contar capa e referências), e deve apresentar configurações adequadas de tamanhos de fonte e margens. O documento deve conter, no mínimo, as seguintes informações:

- Introdução: breve explicação sobre o trabalho e a meta-heurística desenvolvida;
- Problema: descrição clara do problema a ser resolvido, e a formulação dele em Programação Linear, devidamente explicada, ressaltando a utilidade de cada restrição do problema e a função objetivo modelada;
- Descrição **detalhada** do algoritmo proposto: em especial, com as justificativas para as escolhas feitas em cada um dos itens a seguir,
  1. Representação do problema;
  2. Principais estruturas de dados;
  3. Geração da solução inicial;
  4. Vizinhança e a estratégia para seleção dos vizinhos;
  5. Parâmetro(s) do método, com os valores utilizados nos experimentos;
  6. O(s) critério(s) de parada do algoritmo.
- Uma tabela de resultados, com uma linha por instância testada, e com no mínimo as seguintes colunas:
  1. Valor da relaxação linear encontrada pelo GLPK com a formulação matemática;
  2. Valor da melhor solução inteira encontrada pelo GLPK com a formulação matemática (reportar mesmo que não ótima);
  3. Tempo de execução do GLPK (com limite de 1h, ou mais);
  4. Valor médio da solução inicial do seu algoritmo;
  5. Valor médio da melhor solução encontrada pelo seu algoritmo;
  6. Desvio padrão das melhores soluções encontradas pelo seu algoritmo;
  7. Tempo de execução médio, em segundos, do seu algoritmo;
  8. Desvio percentual médio das soluções obtidas pelo seu algoritmo em relação à melhor solução conhecida. Assumindo que  $S$  seja a solução obtida por seu algoritmo e  $S^*$  seja a melhor conhecida, o desvio percentual para problemas de minimização é dado por  $100 \frac{S-S^*}{S^*}$ ; e de maximização por  $100 \frac{S^*-S}{S^*}$ ;

- Análise dos resultados obtidos;
- Conclusões;
- Referências utilizadas.

Os resultados da meta-heurística devem ser a média de 10 execuções (no mínimo) para cada instância. Cada execução deve ser feita com uma *semente* (do inglês, *seed*) de aleatoriedade diferente, a qual deve ser informada para fins de reprodutibilidade (uma sugestão é usar sementes no intervalo  $[1, k]$ , com  $k$  o número de execuções). Note que a semente é um meio de fazer com que o gerador de números aleatórios gere a mesma sequência quando a mesma semente é utilizada <sup>2</sup>.

## 5 Problemas

Esta seção descreve os problemas considerados neste trabalho. Cada grupo deve resolver aquele que escolheu.

### Permutational Flowshop Scheduling Problem (PFSP)

**Definição:** É composta por um conjunto de tarefas  $N = \{1, \dots, n\}$  e um conjunto de máquinas  $M = \{1, \dots, m\}$ . Todas as tarefas devem ser processadas em todas as máquinas, e todas as máquinas devem processar as tarefas em uma mesma sequência. Por exemplo, algumas das sequências possíveis para uma instância com quatro tarefas são (1-2-3-4), (2-1-4-3) e (3-2-1-4). As tarefas exigem um tempo específico para serem processadas por cada máquina, que é definido pelo parâmetro  $p_{ir} \geq 0$ , para cada tarefa  $i \in N$  e máquina  $r \in M$ . As tarefas devem ser processadas sequencialmente no conjunto das máquinas, isto é, uma tarefa  $i \in N$  deve ser processada na máquina 1, então na máquina 2, e assim por diante até a máquina  $m$ .

**Solução:** Ordem de execução das tarefas.

**Objetivo:** Encontrar uma ordenação de tarefas que minimiza o *makespan* relacionado ao tempo de processamento da última máquina.

**Referência base:** Tseng et al. (2003). An empirical analysis of integer programming formulations for the permutation flowshop. [Omega](#).

**Arquivos de instâncias:** disponíveis em [Web of Instances](#). Note que somente as instâncias listadas abaixo devem ser resolvidas. Por questões de disponibilidade, as instâncias também estão disponíveis no [Github](#).

**Melhores valores de solução conhecidos:** A table abaixo lista os *best known values* para as instâncias a serem testadas. Você também pode consultar os BKS no site em que as instâncias estão publicadas. Outra opção é usar os valores do material suplementar de *New hard benchmark for flowshop scheduling problems minimising makespan*, por Vallada, Ruiz e Framinan (2015). [European Journal of Operational Research](#).

<sup>2</sup>Na linguagem C++, por exemplo, a semente é passada para o método construtor da classe `std::mt19937`, que implementa o gerador de números pseudo aleatórios de Mersenne.

Instância	$ N $	$ M $	BKS
VFR10_15_1_Gap	10	15	1307
VFR20_10_3_Gap	20	10	1592
VFR20_20_1_Gap	20	20	2270
VFR60_5_10_Gap	60	5	3663
VFR60_10_3_Gap	60	10	3423
VFR100_60_1_Gap	100	60	9395
VFR500_40_1_Gap	500	40	28548
VFR500_60_3_Gap	500	60	31125
VFR600_20_1_Gap	600	20	31433
VFR700_20_10_Gap	700	20	36417

### Parallel Machines Scheduling Problem

**Definição:** Considere um conjunto de  $N = \{1, \dots, n\}$  de tarefas que devem ser processadas uma única vez por alguma máquina do conjunto  $M = \{1, \dots, m\}$ . Cada tarefa  $i \in N$  requer  $p_{ik} \geq 0$  unidades de tempo para ser processada na máquina  $k \in M$ . As máquinas exigem um tempo de preparo para execução das tarefas, que depende da sequência de tarefas a serem processadas. Dessa forma, uma máquina  $k \in M$  requer  $s_{ijk}$  unidades de tempo para começar o processamento da tarefa  $j \in N$ , tendo  $i \in N$  como tarefa antecessora. O tempo de preparo da máquina para a primeira tarefa é 0. Por questões de simplicidade, pode-se considerar o parâmetro  $G_{ijk} = p_{ik} + s_{ijk}$ ,  $\forall i, j \in N, k \in M$ .

**Solução:** Ordem de atribuição das tarefas às máquinas.

**Objetivo:** Encontrar um escalonamento das tarefas às máquinas que minimiza o *makespan* relacionado a máquina que opera por mais tempo.

**Referência base:** Ezugwu (2019). Enhanced symbiotic organisms search algorithm for unrelated parallel machines manufacturing scheduling with setup times. [Knowledge-Based Systems](#).

**Arquivos de instâncias:** disponíveis em [Scheduling Research Virtual Center](#). Por questões de disponibilidade, as instâncias também estão disponíveis no [Github](#).

**Melhores valores de solução conhecidos:** Disponíveis no site das instâncias ou no artigo referência.

Instância	$ N $	$ M $	BKS
20on4Rp50Rs50_1	20	4	$527.80 \pm 15.43$
60on8Rp50Rs50_1	60	8	$820.00 \pm 9.62$
60on4Rp50Rs50_1	60	4	$1673.20 \pm 43.67$
80on8Rp50Rs50_1	80	8	$1089.00 \pm 7.25$
80on12Rp50Rs50_1	80	12	$711.60 \pm 5.73$
100on2Rp50Rs50_1	100	2	$5872.00 \pm 33.32$
100on6Rp50Rs50_1	100	6	$1858.40 \pm 9.07$
100on8Rp50Rs50_1	100	8	$1371.00 \pm 12.10$
120on12Rp50Rs50_1	120	12	$1087.80 \pm 32.26$
120on10Rp50Rs50_1	120	10	$1326.80 \pm 13.46$

### Traveling Salesman Problem with Draft Limits

**Definição:** Uma embarcação deve ser utilizada para transporte de mercadorias de um porto garagem até um conjunto de portos consumidores. Considere  $V = \{1, \dots, v\}$  o conjunto de portos do problema, em que o porto 1 é considerado o porto garagem. Cada porto consumidor  $j \in V \setminus \{1\}$  tem uma demanda  $d_j$  unidades de consumo, que deve ser atendida pela embarcação em uma única visita. Por questões técnicas, cada porto suporta uma embarcação com profundidade máxima  $l_j$ . De maneira geral, a profundidade da embarcação está diretamente relacionada a quantidade de carga em seu convés. Quanto mais carregado está a embarcação, maior é a parcela do casco da que encontra-se submersa. e vice e versa. Assim, a embarcação só pode atracar um porto no momento em que sua parcela submersa esteja abaixo do limite que o porto suporta. Além disso, deve-se considerar o parâmetro  $c_{ij}$  que indica o custo para deslocamento da embarcação entre os pares de portos  $(i, j) \in V \times V$ .

**Solução:** Ordem de visitação dos portos, observando aos limites de profundidades dos portos consumidores.

**Objetivo:** Rota de menor custo.

**Referência base:** Rakke et al. (2012). The Traveling Salesman Problem with Draft Limits. [Computers & Operations Research](#).

**Arquivos de instâncias:** disponíveis na [Biblioteca do TSPDL](#). Por questões de disponibilidade, as instâncias também estão disponíveis no [Github](#).

**Melhores valores de solução conhecidos:** Alguns estão publicados no artigo de referência, e outros em Todosijević et al. (2017). A general variable neighborhood search variants for the travelling salesman problem with draft limits. [Optimization Letters](#). Valores marcados em negrito referem-se às soluções ótimas para as referidas instâncias.

Instância	BKS (média)
bayg29_10_1	<b>1713.60</b>
bayg29_50_1	<b>2091.00</b>
gr17_25_1	<b>2237.70</b>
gr48_10_1	<b>6635.70</b>
gr48_25_1	<b>5800.30</b>
KroA200_50_1	30665.20
KroA200_75_1	30896.10
pcb442_50_1	59858.30
pcb442_75_1	61010.10
Ulysse_22_50_1	<b>8425.60</b>

## 6 Material auxiliar

Algumas referências auxiliares sobre as meta-heurísticas:

1. Simulated Annealing: 'Optimization by simulated annealing, by Kirkpatrick, Scott, C. Daniel Gelatt, and Mario P. Vecchi. Science 220.4598 (1983): 671-680'. (<http://leonidzhukov.net/hse/2013/stochmod/papers/KirkpatrickGelattVecchi83.pdf>)
2. Tabu Search: 'Tabu Search: A Tutorial, by Fred Glover (1990), Interfaces.' (<http://leeds-faculty.colorado.edu/glover/Publications/TS%20-%20Interfaces%20Tutorial%201990%20aw.pdf>)
3. Genetic Algorithm: 'A genetic algorithm tutorial, by D. Whitley, Statistics and computing 4 (2), 65-85.' (<http://link.springer.com/content/pdf/10.1007%252FBF00175354.pdf>)
4. GRASP: 'T.A. Feo, M.G.C. Resende, and S.H. Smith, A greedy randomized adaptive search procedure for maximum independent set, Operations Research, vol. 42, pp. 860-878, 1994' (<http://mauricio.resende.info/doc/gmis.pdf>).
5. VNS: 'A Tutorial on Variable Neighborhood Search, by Pierre Hansen (GERAD and HEC Montreal) and Nenad Mladenovic (GERAD and Mathematical Institute, SANU, Belgrade), 2003.' (<http://www.cs.uleth.ca/~benkoczi/OR/read/vns-tutorial.pdf>)
6. ILS: 'Iterated local search. Lourenço, Helena R., Olivier C. Martin, and Thomas Stutzle. International series in operations research and management science (2003): 321-354.' (<https://arxiv.org/pdf/math/0102188.pdf>).

## 7 Dicas

A seguir algumas dicas gerais para o trabalho:

1. Uma boa vizinhança é aquela que permite alcançar qualquer solução do problema, dado um número suficiente iterações (a vizinhança imediata de uma determinada solução **não deve** conter todas as possíveis soluções);
2. No caso de vizinhos com mesmo valor de solução, utilizar escolhas aleatórias como critério de desempate pode trazer bons resultados (pode-se usar *reservoir sampling*, veja [aqui](#));

3. É importante **analisar bem** a complexidade assintótica das operações do algoritmo, considerando as estruturas de dados escolhidas e as vizinhanças usadas;
4. Para mais informações e dicas sobre experimentos com algoritmos: Johnson, David S. “A theoretician’s guide to the experimental analysis of algorithms.” (2001). (<https://www.cc.gatech.edu/~bader/COURSES/GATECH/CSE6140-Fall2007/papers/Joh01.pdf>);
5. Veja a apresentação do Professor Marcus Ritt sobre “como perder pontos” (disponível em <https://www.inf.ufrgs.br/~MRPRITT/lib/exe/fetch.php?media=inf05010:tp20171.pdf>);
6. Ainda de autoria do Professor Marcus Ritt, há um *cheat sheet* interessante sobre GLPK e MathProg (disponível em <https://www.inf.ufrgs.br/~mrpritt/lib/exe/fetch.php?media=inf05010:glpk-quickref.pdf>);
7. Em caso de dúvidas, não hesitem em contatar a Prof. Luciana Buriol ou o Doutorando Alberto Kummer (Lab. 207 do prédio 43424 - [alberto.kummer@gmail.com](mailto:alberto.kummer@gmail.com))