Trabalho Final de Otimização Combinatória (2018/2)

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, 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;
- Todas as implementações devem fazer a leitura de uma instância no formato do problema na entrada padrão (`stdin`) e imprimir a melhor solução encontrada, bem como o tempo de execução, na saída padrão (`stdout`);

- Os parâmetros do método de resolução devem ser **recebidos via linha de comando**[1], em especial a semente para o gerador de números pseudo aleatórios;
- 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 incumbência 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;

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

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 [2].

5 Problemas

Esta seção descreve os problemas considerados neste trabalho. Cada grupo deve resolver aquele que escolheu.
Dica: utilize o DOI (`https://www.doi.org/`) para obter os trabalhos de referência dos problemas!

Multiple Depot Vehicle Scheduling Problem (MDVSP)

Instância: é composta por um conjunto $K = \{1, \ldots, k\}$ de garagens e um conjunto $T = \{1, \ldots, t\}$ de viagens. Considerando $P = \{1, \ldots, k, k+1, \ldots, k+t\}$, a matriz de adjacências $M_{i,j}$ indica a viabilidade de execução dos pares de viagens $i, j \in P$. Cada posição da matriz apresenta um valor de custo associado, ou -1 denotando uma sequência de viagens infactível. Além disso, cada garagem $k \in K$ apresenta um número máximo de veículos disponíveis $v_k \in \mathbb{N}^*$.

Solução: diversas sequências de viagens (caminhos). Cada veículo sai de uma das garagens do problema, executa uma sequência de viagens compatíveis, e retorna para a sua respectiva garagem.

Objetivo: Dada uma matriz de conexões $M_{i,j}$ e a capacidade das garagens, encontre a solução de menor custo em que cada viagem de T é executada por exatamente um dos veículos da solução. O número de veículos utilizados não deve ultrapassar a capacidade das garagens do problema.

Referência base: Pepin et al. (2009). A comparison of five heuristics for the multiple depot vehicle scheduling problem. Journal of Scheduling. DOI `10.1007/s10951-008-0072-x`.

Arquivos de instâncias: disponíveis em `https://personal.eur.nl/huisman/instances.htm`. Note que somente as instâncias listadas abaixo devem ser resolvidas.

Melhores valores de solução conhecidos: Você também pode consultar os BKS no site em que as instâncias estão publicadas. Valores marcados em negrito

[2] 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

referem-se às soluções ótimas para as referidas instâncias.

Instância	Núm. garagens	Núm. viagens	BKS	
			Valor obj.	Núm. veículos
m4n500s2	4	500	**1283811**	123
m4n500s4	4	500	**1317077**	126
m8n500s0	8	500	1292411	124
m4n1000s3	4	1000	2490812	237
m4n1000s4	4	1000	**2519191**	238
m8n1000s0	8	1000	2422112	232
m8n1000s2	8	1000	2556313	247
m4n1500s1	4	1500	3559176	338
m8n1500s1	8	1500	3802650	366
m8n1500s4	8	1500	3704953	360

Weighted Tardiness

Instância: Considere um conjunto de tarefas $N = \{1, \ldots, j\}$ que devem ser processadas sequencialmente por uma máquina. Interrupções não são permitidas, e uma vez iniciado o processamento, uma tarefa devem ser executada integralmente. Todas as tarefas estão disponíveis para processamento no tempo 0, e cada tarefa $j \in N$ demanda de um tempo de processamento positivo p_j. Além disso, cada tarefa possui um peso w_j e um tempo máximo ideal para entrega d_j.

Solução: ordem de execução das tarefas. Considere c_j o tempo em que a tarefa $j \in N$ foi finalizada. Assim, o atraso para finalização da tarefa $j \in N$ é dado por $t_j = \max\{c_j - d_j, 0\}$ (t_j vale 0 caso a tarefa seja entregue **até** o tempo máximo ideal de finalização da tarefa j).

Objetivo: todas as tarefas devem ser executadas pela máquina, e deve-se escolher a ordem de execução que minimize a expressão $\sum_{j \in N} w_j t_j$.

Referência base: Congram et al. (2002). An Iterated Dynasearch Algorithm for the Single-Machine Total Weighted Tardiness Scheduling Problem. INFORMS Journal on Computing. DOI `10.1287/ijoc.14.1.52.7712`.

Arquivos de instâncias: disponíveis em `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html`. Cada um dos arquivos de instância possui 125 casos de teste. Note que somente as instâncias e casos de teste listados abaixo devem ser resolvidas.

Melhores valores de solução conhecidos: Você também pode obter os melhores valores de solução conhecidos dos arquivos `wtopt40.txt`, `wtopt50.txt`, `wtbest100a.txt` e `wtbest100b.txt`, disponíveis juntamente aos arquivos de instância. Valores marcados em negrito referem-se às soluções ótimas para as referidas instâncias.

Arquivo	N	Caso de teste	BKS
wt40	40	9	**16225**
wt40	40	28	**15**
wt40	40	105	**0**
wt50	50	41	**71111**
wt50	50	77	**0**
wt50	50	119	**106043**
wt100	100	26	8
wt100	100	30	50
wt100	100	46	829828
wt100	100	80	**0**

Aircraft landing

Instância: Considere um conjunto de aeronaves $P = \{1, \ldots, i\}$ que precisam aterrisar em um aeroporto. Cada uma das aeronaves $i \in P$ possui um horário mínimo (E_i), máximo (L_i) e ideal (T_i) de aterrisagem. Há um custo g_i para cada unidade de tempo em que a aeronave pousa **antes** do horário ideal. De maneira semelhante, há um custo h_i para cada unidade de tempo que a aeronave pousa **após** o horário ideal. Por questões de segurança, há um tempo de espera entre o pouso das aeronaves $i, j \in P$, definido pelo parâmetro S_{ij} (unidades de tempo). Esse tempo de espera é definido entre todos os pares de aeronaves do problema.

Solução: ordem de aterrisagem das aeronaves. Todas as aeronaves devem pousar conforme seus horizontes de tempo mínimo e máximo de aterrisagem, respeitando o tempo de separação entre o pouso das aeronaves.

Objetivo: Considere x_i o tempo em que a aeronave $i \in P$ aterrisa no aeroporto. Busca-se uma solução de menor custo que minimiza os atrasos e adiantamento dos pousos das aeronaves, conforme a expressão $\sum_{i \in P} g_i \max\{T_i - x_i, 0\} + h_i \max\{x_i - T_i\}$.

Referência base: Beasley et al. (2004). Displacement problem and dynamically scheduling aircraft landings. Journal of the Operational Research Society. DOI 10.1057/palgrave.jors.2601650.

Arquivos de instâncias: disponíveis em http://people.brunel.ac.uk/~mastjjb/jeb/orlib/airlandinfo.html. Note que somente as instâncias listadas abaixo devem ser resolvidas. Para todos os casos de teste, considere a existência de uma pista de pouso apenas.

Melhores valores de solução conhecidos: Você também pode obter os melhores valores de solução conhecidos das Tabelas 1 e 2 da referência base. Valores marcados em negrito referem-se às soluções ótimas para as referidas instâncias.

Instância	Núm. aeronaves	BKS
airland1	10	**700**
airland2	20	**820**
airland4	20	**2520**
airland5	20	**3100**
airland6	30	**24442**
airland7	44	**1550**
airland8	50	**1950**
airland11	200	66427.28
airland12	250	81916.40
airland13	500	178725.16

6 Material auxilar

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*);
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. 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);