Trabalho Final de Otimização Combinatória (2019/2)

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 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;
- 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 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;

[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.

Mirrored Traveling Tournament Problem (mTTP)

Definição: O *Mirrored Traveling Tournment Problem* consiste em organizar a agenda das partidas de um torneio esportivo. As partidas do torneio ocorrem em diversas cidades, e cada equipe participante possui uma cidade-natal no quais acontecem os jogos "em casa". Dessa forma, o mTTP consiste em minimizar a distância total percorrida pelas n equipes participantes, observando uma série de restrições práticas do problema. O campeonato é organizado em dois turnos, e cada par de equipes se enfrenta exatamente uma vez por turno. Assim, cada turno contém $(n-1)$ rodadas. Os dois turnos são espelhados, no sentido que as rodadas do primeiro turno são as mesmas do segundo, com os locais de partida sendo invertidos (se no primeiro turno a equipe E_1 enfrenta E_2 em casa, então E_1 enfrenta E_2 fora de casa no segundo turno). Cada equipe só participa de um confronto por rodada, e nenhuma equipe deve participar de 3 ou mais confrontos "em casa" ou "fora de casa" em sequência.

Solução: Agenda dos confrontos do torneio.

Objetivo: Minimizar a distância total percorrida pelas equipes para a execução do torneio esportivo.

Referência base: Santos e Carvalho. (2018). Algoritmo Genético Aplicado à Otimização do Planejamento de Torneios Esportivos. Anais do 50º SBPO. Carvalho e Lorena. (2012). *New models for the Mirrored Traveling Tournament Problem*. Computers & Industrial Engineering.

Arquivos de instâncias: disponíveis na biblioteca do Challenge Traveling Tournament Instances. Note que somente as instâncias listadas abaixo devem ser

[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.

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.

Instância	BKS
Nl4	8276
Nl6	26588
Nl8	41928
Nl10	63832
Nl12	119608
Nl14	199363
circ6	72
circ8	140
circ10	272
circ12	432

Maximally Diverse Grouping Problem (MDGP)

Definição: Busca-se agrupar $i = 1, \ldots, n$ pessoas em $g = 1, \ldots, m$ grupos de trabalho, de maneira que cada pessoa esteja vinculada a exatamente a um grupo, e que cada grupo tenha um entre a_g e b_g participantes. Para cada par de pessoas, é conhecido um valor d_{ij} que representa o quão diferente são as pessoas i e j.

Solução: Alocação das pessoas aos grupos.

Objetivo: Maximizar a soma da diversidade dos grupos.

Referência base: Araujo e Figueiredo (2018). O problema da diversidade máxima de grupos: uma abordagem de programação linear inteira. Anais do 50º SBPO.

Arquivos de instâncias: disponíveis na MDGPLIB. Por questões de disponibilidade, as instâncias também estão disponíveis no Github.

Melhores valores de solução conhecidos: Disponíveis no artigo referência. Valores marcados em negrito referem-se às soluções ótimas para as referidas instâncias.

Instância	n	Tam. dos grupos	BKS
RanInt_07	10	$\neq$	**1221.00**
RanReal_10	10	=	**1195.92**
RanInt_03	12	=	**993.00**
Geo_10	10	=	**3752.03**
RanInt_10	10	=	**1112.00**
RanInt_03	60	$\neq$	17041
RanReal_04	60	=	18050.80
RanInt_05	30	=	5496.00
Geo_04	60	=	45971.80
Geo_08	30	$\neq$	13282.00

The home health care routing and scheduling problem with interdependent services (HHCRSP)

Definição: Considere um conjunto de veículos $\mathcal{V}$ e um conjunto de pontos georreferenciados $\mathcal{C}$ que representam pacientes de um plano de saúde. O conjunto $\mathcal{S}$ representa todas as especialidades médicas consideradas no problema. Os pacientes possuem demandas de diversas especialidades médicas para atendimento domiciliar, e cada demanda leva um certo tempo visitação para ser cumprida. Além disso, os pacientes só podem receber visitas em uma certa faixa de horários do dia, com o horário de início das visitas devendo ser estritamente cumpridos. Considera-se que cada veículo do problema possui uma equipe de profissionais de saúde vinculados a ele. Naturalmente, um veículo só é capaz de atender algumas especialidades médicas. Os veículos devem partir de uma unidade de saúde central para cumprir o atendimento de todos os pacientes. No final dos atendimentos, todas as equipes devem retornar à unidade central de saúde.

Solução: Para cada veículo, uma rota com a ordem de visitação dos pacientes.

Objetivo: Encontrar uma solução cuja a soma das distâncias percorridas pelos veículos, dos atendimentos fora da faixa de horário, e a maior extrapolação do horário de visita sejam o menor possível. **Desconsiderar as restrições de sincronização (11) e (12)!**

Referência base: Mankowska et al. (2014). *The home health care routing and scheduling problem with interdependent services*. Health Care Management Science.

Arquivos de instâncias: disponíveis em Benchmark instances for the HHCRSP. Por questões de disponibilidade, as instâncias também estão disponíveis no Github.

Limitantes superiores: Neto et al. (2019). *A Matheuristic Algorithm Applied to the Home Health Care Problem*. Anais do 51º SBPO.

Instância	Limitante Superior
A3	305.90
B3	399.20
B9	403.80
C1	1006.72
C2	597.06
C6	852.04
D10	1306.60
E8	832.73
F1	1721.40
G9	2415.50

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*, 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)