Trabalho Final de Otimização Combinatória (2020/1)

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**[1], em especial a semente para o gerador de números pseudo aleatórios;

[1] 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 [2].

5 Problemas

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

Problema da coloração de vértices

Instância: As instância desse problemas são grafos definidos como $G = \{V, E\}$, onde V é conjunto de vértices do grafo e E o conjunto de arestas.

Solução: Uma solução para esse problema é a atribuição de cores para os vértices tal que dois vértices ligado por uma aresta não tenham a mesma cor.

Objetivo: Achar a menor quantidade de rotulos (cores) que respeite a restrição.

Referência base: Malaguti et al. (2011), An exact approach for the Vertex Coloring Problem (Revista Discrete Optimization).

Arquivos de instâncias: Disponíveis em Graph Coloring Benchmarks. Por questões de disponibilidade, as instâncias também estão disponíveis em Github.

Melhores valores de solução conhecidos: Os melhores valores conhecidos podem ser encontrado em Malaguti et al. (2011), An exact approach for the Vertex Coloring Problem (Revista Discrete Optimization).

Instância	|V|	|E|	MVC
2-FullIns_4	212	1621	6
4-FullIns_3	114	541	7
5-FullIns_3	154	792	3
queen10_10	100	2940	11
queen11_11	121	3960	11
queen12_12	244	5192	12
queen13_13	169	6656	13
queen14_14	196	4186	14
queen15_15	225	5180	15
queen16_16	256	12640	17

[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.

Problema da menor arvore geradora rotulada

Instância: As instâncias desse problemas são grafos rotulados em arestas, esses grafos são definidos por um conjunto V de vértices, E de arestas e um conjunto K de rótulos.

Solução: Uma solução para esse problema é uma árvore geradora.

Objetivo: Encontrar uma árvore geradora que minimize o número de cores.

Referência base: Krumke e Wirth (1998). On the minimum label spanning tree problem. (Revista Information Processing Letters)

Arquivos de instâncias: As istância desse problema foram gerada para a cadeira de Otimização combinatória usando como referência as definições de Cerulli et al (2014) "The k-labeled spanning forest".(Procedia - Social and Behavioral Sciences). Elas estão disponiveis em: Github, assim como uma breve descrição de como elas foram geradas.

Melhores valores de solução conhecidos: A tabela abaixo apresenta os melhores valores conhecidos para as instâncias do trabalho, sendo valores com $*$ valores otimos encontrado pelo solver.

Instância	$\|V\|$	$\|E\|$	$\|L\|$	BKS
testFile_0_10_5	10	9	5	5*
testFile_7_75_37	75	555	37	10
testFile_8_75_37	75	555	37	5*
testFile_9_75_60	75	555	60	7*
testFile_10_75_60	75	555	60	12
testFile_11_75_93	75	555	75	20
testFile_12_75_93	75	555	75	19
cerulli_100_25	100	990	25	-
cerulli_100_50	100	990	50	-
cerulli_100_100	100	990	100	-

Problema da floresta geradora mínima k-rotulada (kLSF)

Instância: As instâncias desse problemas são grafos rotulados em arestas, esses grafos são definidos por um conjunto V de vértices, E de arestas e um conjunto K de rótulos, além de um valor inteiro $0 \geq k_{max} \geq k$.

Solução: Uma floresta geradora onde as arestas desta floresta pertencem a no máximo k_{max} cores.

Objetivo: Encontrar uma floresta que minimize o número de árvores (componentes).

Referência base: Cerulli et al (2014) "The k-labeled spanning forest". (Procedia - Social and Behavioral Sciences)

Arquivos de instâncias: As istância desse problema foram gerada para a cadeira de Otimização combinatória usando como referência as definições de Cerulli et al (2014) "The k-labeled spanning forest".(Procedia - Social and Behavioral Sciences). Elas estão disponiveis em: Github, assim como uma breve

descrição de como elas foram geradas.

Melhores valores de solução conhecidos: A tabela abaixo apresenta os melhores valores conhecidos para as instâncias do trabalho, sendo valores com $*$ valores otimos encontrado pelo solver.

Instância	$\|V\|$	$\|E\|$	$\|L\|$	k_max	BKS
testFile_0_10_5	10	9	5	3	4*
testFile_1_50_25	75	555	37	4	4*
testFile_4_50_40	75	555	37	5	8
testFile_6_50_62	75	555	60	6	13
testFile_7_75_37	75	555	60	4	7
testFile_9_75_60	75	555	75	5	16
testFile_11_75_93	75	555	75	6	22
cerulli_100_25	100	990	25	3	-
cerulli_100_50	100	990	50	6	-
cerulli_100_100	100	990	100	6	-

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);