Trabalho Final de Otimização Combinatória (2018/1)

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 deste problema. O trabalho deve ser realizado em grupos. Cada grupo escolhe uma combinação em forma de par (*problema* , *meta-heurística*). Um par não pode ser repetido por outro grupo.

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 (`.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 `C`, `C++`, `Java` ou `Julia`, sem o uso de bibliotecas proprietárias, e podendo ser compilada e executada pelo menos em ambiente Linux ou Windows. Além disso, alguns critérios básicos devem ser levados em conta no desenvolvimento:

- Critérios de engenharia de software: documentação e legibilidade;
- Todas as implementações devem receber 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 devem ser recebidos via linha de comando[1], em especial a semente de aleatoriedade;

[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 com o vizinho, e não toda a solução novamente).
- O critério de parada do algoritmo *não* deve ser tempo de execução. Alguns exemplos possíveis: 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 *dois 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 (i.e., `boost`).

4 Relatório

O *relatório*, a ser entregue em formato PDF, deve possuir no máximo seis páginas (sem contar capa e referências, e com configurações adequadas de tamanhos de fonte e margens), e 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;
- 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 melhor solução encontrada pelo GLPK com a formulação matemática (reportar mesmo que não ótima);
  2. Tempo de execução do GLPK (com limite de 1h, ou mais);
  3. Valor médio da solução inicial do seu algoritmo;
  4. Valor médio da melhor solução encontrada pelo seu algoritmo;
  5. Desvio padrão das melhores soluções encontradas pelo seu algoritmo;
  6. Tempo de execução médio (em segundos) do algoritmo;
  7. 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^* a melhor conhecida, o desvio 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.

Simple Assembly Line Balancing Problem type 1 (SALBP-1)

Instância: é composta por um conjunto N de tarefas, um conjunto S de estações de trabalho, um grafo de precedência de tarefas $(i, j) \in P$, um tempo t_w de cada tarefa $w \in N$ e um tempo de ciclo c.

Solução: uma sequência de tarefas $w \in N$ que opera em cada estação $s \in S$, tal que cada tarefa deve ser executada apenas uma vez e em apenas uma estação considerando as regras de precedência e o tempo de ciclo limite da estação.

Objetivo: Dado um tempo de ciclo c, deve-se minimizar o número total de estações m utilizadas.

Referência base: Marcus Ritt and Alysson M. Costa. A comparison of formulations for the simple assembly line balancing problem. Int. Trans. Oper. Res., 2015.

Arquivos de instâncias: disponíveis em
`http://inf.ufrgs.br/~vsportella/`

Melhores valores conhecidos: (BKS)

Instância	c	BKS (valor de m)
Mertens	6	6
Mertens	15	2
Mertens	18	2
Mitchell	14	8
Mitchell	35	3
Mitchell	26	5
Wee-mag	56	30
Wee-mag	28	63
Wee-mag	36	60
Wee-mag	54	31

[2] Na linguagem C, por exemplo, a semente é passada para a função `srand()` no início de um programa.

Simple Assembly Line Balancing Problem type 2 (SALBP-2)

Instância: é composta por um conjunto N de tarefas, um conjunto S de estações de trabalho, um grafo de precedência de tarefas $(i, j) \in P$, um tempo t_w de cada tarefa $w \in N$.

Solução: uma sequência de tarefas $w \in N$ que opera em cada estação $s \in S$, tal que cada tarefa deve ser executada apenas uma vez e em apenas uma estação considerando as regras de precedência.

Objetivo: Dado um número de estações m, deve-se minimizar o tempo de ciclo c.

Referência base: Marcus Ritt and Alysson M. Costa. A comparison of formulations for the simple assembly line balancing problem. Int. Trans. Oper. Res., 2015.

Arquivos de instâncias: disponíveis em
`http://inf.ufrgs.br/~vsportella/`

Melhores valores conhecidos: (BKS)

Instância	m	BKS (valor de c)
Hahn	3	4787
Hahn	8	1907
Hahn	10	1775
Lutz3	3	548
Lutz3	20	85
Lutz3	5	329
Wee-mag	3	500
Wee-mag	10	150
Wee-mag	20	77
Wee-mag	30	56

Generalized Assignment Problem (GAP)

Instância: é composta por um conjunto de agentes I e tarefas J. Para cada $i \in I$ e $j \in J$ existe um custo c_{ij} da tarefa j ser atribuída ao agente i, um recurso a_{ij} requerido pelo agente i para realizar a tarefa j e um valor b_i de recursos disponíveis do agente i.

Solução: uma sequência de tarefas j atribuída a cada agente i, desde que todas as tarefas sejam executadas exatamente uma vez e os limites dos recursos de cada agente seja respeitado.

Objetivo: minimizar os custos.

Referência base: P C Chu and J E Beasley "A genetic algorithm for the generalised assignment problem", Computers and Operations Research, Volume 24, pages 17-23, 1997.

Arquivos de instâncias: disponíveis em
`http://inf.ufrgs.br/~vsportella/`

Melhores valores conhecidos: (BKS)

Instância	BKS
gapa-1	1698
gapa-2	3235
gapa-3	1360
gapa-4	2623
gapa-5	1158
gapa-6	2339
gapd-1	6353
gapd-6	12266 (0.35%)
gapd-2	12743 (0.02%)

Informações adicionais sobre os dados das instâncias se encontram em um arquivo `readme` dentro de cada um dos pacotes compactados fornecidos. É muito importante que leiam completamente as informações presentes neste arquivo.

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 dadas suficientes iterações (a vizinhança imediata de uma determinada solução não deve conter todas as possíveis soluçoes);
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." Data Structures, Near Neighbor Searches, and Methodology (1999). (`http://www.ifs.tuwien.ac.at/~weippl/johnson.pdf`);
5. Em caso de dúvidas, não hesitem em contatar a Prof. Luciana Buriol, ou a Mestranda Victória Simonetti (Lab. 207 do prédio 43424 - `victoriasimonetti@gmail.com`);