Trabalho Final de Otimização Combinatória (2017/2)

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 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.

Orienteering Problem (OP)

Instância: é composta por um grafo completo $G = (V, A)$, um vértice de origem $v_0 \in V$, e um custo máximo C. Cada vértice $v \in V$ possui uma pontuação associada p_v. Cada aresta $a \in A$ possui um custo associado d_a.

Solução: uma rota $R = \{v_0, v_1, \ldots, v_0\}$, iniciando e terminando em v_0, tal que cada vértice $v \in V \setminus \{v_0\}$ aparece no máximo uma vez em R, e o custo desse caminho não seja maior que C.

Objetivo: encontrar a rota R de maior soma de pontuações p_v dos vértices visitados.

Referência base: Dominique Feillet, Pierre Dejax, and Michel Gendreau. Traveling Salesman Problems with Profits. Transportation Science 39, 2 (2005), 188-205.

Arquivos de instâncias: disponíveis em
`http://inf.ufrgs.br/~cssartori/ta/op_inst.zip`

Melhores valores conhecidos: (BKS)

Instância	BKS	Instância	BKS
a8	190	a16	245
ber25	609	bier127	2365
eil51	1399	eil76	2467
kroA200	6123	lin105	2986
pr152	3905	rat99	2908

Travelling Salesman Problem with Time Windows (TSPTW)

Instância: é composta por um grafo completo $G = (V, A)$ e um vértice de origem $v_0 \in V$. Cada vértice $v \in V$ possui uma janela de tempo associada, da forma $[a_v, b_v]$, indicando que o vértice deve ser visitado dentro deste período. Cada aresta $a \in A$ possui um tempo de viagem associado t_a.

Solução: uma rota $R = \{v_0, v_1, \ldots, v_0\}$, iniciando e terminando em v_0, tal que todos os vértices são visitados ao longo da rota R uma única vez, e o tempo t_v no qual um vértice v é visitado no caminho seja $a_v \leq t_v \leq b_v$.

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

Objetivo: encontrar a rota R de menor soma dos tempos das arestas percorridas.

Referência base: Desrosiers, Jacques, et al. "Time constrained routing and scheduling."Handbooks in operations research and management science 8 (1995): 35-139.

Arquivos de instâncias: disponíveis em
`http://inf.ufrgs.br/~cssartori/ta/tsptw_inst.zip`

Melhores valores conhecidos: (BKS)

Instância	BKS	Instância	BKS
n10w140	206	n20w140	176
n40w140	328	n60w140	423
n80w140-1	512	n80w140-2	472
n100w140-1	604	n100w140-2	615
n150w140	762	n200w140	834

Sequential Ordering Problem (SOP)

Instância: é composta por um grafo direcionado $G = (V, A)$, um vértice de origem $v_0 \in V$, um vértice final $v_f \in V$ e pares de precedência $(v_i, v_j), v_i, v_j \in V$. Cada par de precedência (v_i, v_j) indica que o vértice v_i só pode ser visitado em algum momento depois do vértice v_j ter sido visitado. Cada aresta $a \in A$ possui um custo associado d_a.

Solução: um caminho $P = \{v_0, v_1, \ldots, v_k, v_f\}$, iniciando em v_0 e terminando em v_f, passando por todos os vértices exatamente uma vez, e respeitando as relações de precedência definidas na instância.

Objetivo: encontrar o caminho P de menor custo.

Referência base: Gambardella L.M , Dorigo M. An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem, INFORMS Journal on Computing, vol.12(3), pp. 237-255, 2000.

Arquivos de instâncias: disponíveis em
`http://inf.ufrgs.br/~cssartori/ta/sop_inst.zip`

Melhores valores conhecidos: (BKS)

Instância	BKS	Instância	BKS
esc07	2125	esc12	1675
esc25	1681	esc47	1288
esc78	18230	ft70.1	39313
prob.100	1163	rbg109a	1038
rbg150a	1750	rbg174a	2033

Informações adicionais sobre os dados das instâncias se encontram em um arquivo `readme` dentro de cada um dos pacotes compactados fornecidos. As informações incluem: como ler, o que significa cada valor, e como calcular pesos e distâncias dos grafos. 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. Os problemas considerados são variações do TSP, o qual possui diversos tipos de formulações. Algumas contêm um número *exponencial* de restrições, e não são possíveis de serem implementadas diretamente no GLPK (é impraticável gerar todas elas). Para as formulações que exigem restrições adicionais para eliminação de subciclos, pode-se utilizar as restrições de *Miller-Tucker-Zemlin*;
5. 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);
6. Em caso de dúvidas, não hesitem em contatar a Prof. Luciana Buriol, ou o Mestrando Carlo Sartori (Lab. 207 do prédio 43424 - carlosartori2@gmail.com);