Trabalho final de Otimização Combinatória (INF5010) 2017/1

Prof. Dr. Luciana S. Buriol[1], **Henrique Becker**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15064 – CEP 91501-970 – Porto Alegre – RS – Brasil

1. Introdução

O tema do trabalho final da matéria de *Otimização Combinatória* (INF5010) do semestre de 2017/1 é a aplicação de uma *meta-heurística* sobre um *problema NP-completo*. O trabalho consiste em quatro entregáveis: uma *formulação matemática* do problema NP-completo escolhido na linguagem GNU MathProg usada pelo GLPK (ou em Julia), uma *implementação* (em qualquer linguagem) que faça uso da meta-heurística escolhida para resolver o problema NP-completo escolhido, um *relatório* e uma *apresentação*. As exigências relacionadas a esses quatro entregáveis são detalhadas na Seção 2.

O trabalho deve ser realizado em grupo. Cada grupo de alunos escolhe uma entre as cinco meta-heurísticas vistas em aula e um entre os três problemas NP-completos descritos nesse documento. Quando um grupo escolhe uma determinada combinação heurística/problema esta combinação *deixa de estar disponível* para os demais grupos, ou seja, não serão permitidos dois grupos fazerem o trabalho sobre a exata mesma combinação heurística/problema. A descrição das cinco meta-heurísticas permitidas no trabalho (*Simulated Annealing*, Lista Tabu, Programação Genética, GRASP e VNS) foi vista em aula e se encontra disponível no material da disciplina; a Seção 4 traz uma sugestão de material extra externo para cada meta-heurística. A descrição dos problemas NP-completos permitidos no trabalho se encontra na Seção 5, bem como sugestões de instâncias para serem usadas nos experimentos.

2. Exigências do trabalho

Os grupos devem entregar (no moodle da disciplina) um pacote (tar, tar.gz, tar.bz2, zip, rar, ou 7z) contendo *uma única pasta* que por sua vez contém: o relatório em PDF; os slides da apresentação na versão final (igual a que será apresentada) também em PDF; os códigos-fonte de implementação do problema na linguagem de sua escolha; um README(.txt) descrevendo como compilar o código fonte (se é necessário instalar quaisquer bibliotecas, ou se é somente necessário chamar 'make' no terminal) e os parâmetros de linha de comando do binário compilado; a formulação matemática do problema para GNU MathProg (arquivo com extensão "*.mod*"); os arquivos de instância do problema convertidos para entrada de dados do GLPK (arquivos com extensão "*.dat*").

As seguintes subseções descrevem o que os avaliadores esperam de cada um dos três entregáveis do trabalho. O trabalho será corrigido pela Prof. Dr. Luciana S. Buriol e pelo aluno de doutorado Henrique Becker, e uma nota consenso entre ambos será definida.

2.1. Formulação matemática

O entregável *formulação matemática* consiste na formulação matemática do problema para GNU MathProg (arquivo com extensão "*.mod*"), e nos arquivos de instância do problema convertidos para entrada de dados do GLPK (o GLPK permite receber a seção

"data" separadamente, por meio da flag "-d"). Os resultados da execução sobre instâncias do problema deverão ser descritas e justificadas no *relatório* e *apresentação*, e contam pontos para estes entregáveis.

A *formulação matemática* deverá estar *correta*, *legível*, e *comentada*.

2.2. Implementação

O entregável *implementação* consiste nos códigos fontes de um programa que faz uso da meta-heurística escolhida para resolver o problema NP-completo escolhido. As *decisões tomadas na implementação* e os *resultados da execução da mesma sobre instâncias do problema* deverão ser descritas e justificadas no *relatório* e *apresentação*, e contam pontos para esses entregáveis.

Segue a lista de exigências e critérios de avaliação relacionadas a implementação entregue:

- **Não** enviar binários.
- O código fonte deve ser legível e possuir comentários/documentação (este será examinado pelos avaliadores).
- A eficiência das implementações e a qualidade das soluções encontradas por estas serão considerados na avaliação.
- O código fonte não deve depender de funcionalidades específicas de plataforma para compilar ou executar (isto é, deve usar somente o que é descrito no padrão oficial da linguagem utilizada e ser possível de executar em Windows, Mac e Linux).
- O binário compilado a partir do código-fonte deve receber uma instância no formato do problema escolhido[1] na entrada padrão (stdin) e imprimir a melhor solução encontrada, bem como o tempo utilizado na saída padrão (stdout).
- O binário compilado a partir do código-fonte deve receber os principais parâmetros da meta-heurística escolhida por linha de comando (argc e argv em C/C++). Um parâmetro que todas as implementações (independente da meta-heurística) devem ser capazes de receber pela linha de comando é a semente de aleatoriedade a ser usada pela meta-heurística e/ou a solução construtiva inicial. A semente de aleatóriedade *não* pode ser *hardcoded* (fixada no código), muito menos obtida com o uso de `time()` ou qualquer outra função que retorna a data/hora atual.
- O critério de parada da heurística não pode ser tempo. Exemplos de critério de parada possíveis são: número de iterações; número de iterações sem encontrar uma solução melhor do que a melhor encontrada até o momento; proximidade com algum limite superior; ou mais de uma das anteriores (a primeira que ocorrer interrompe o programa). A heurística deve executar, preferencialmente, em poucos segundos. Evitar critério de parada que demande tempo de execução maior do que dois minutos para alguma instância. Resultados com 5 minutos podem ser apresentados se forem relevantes e se resultados com menos de dois minutos forem apresentados conjuntamente. Vale reforçar que o controle do tempo da heurística deve ser feito por meio do ajuste desses critérios de parada, e não pela verificação do número de segundos transcorridos desde o começo da execução do programa,

[1] O formato de cada um dos problemas permitidos é descrito na seção desse documento que aborda o problema específico.

pois isso torna os resultados dependentes do poder de processamento da máquina em que a heurística foi executada.[2]

Algumas dicas:

- Acerca da escolha de vizinhanças: 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 soluções).
- No caso de vizinhos com mesmo valor de solução, utilizar escolhas aleatórias como critério de desempate podem trazer bons resultados.

3. Relatório

O entregável *relatório* consiste em um relatório (arquivo PDF) descrevendo e justificando as decisões tomadas na *implementação*, além dos resultados da execução da *formulação matemática* e da *implementação* sobre as instâncias do problema escolhido. O relatório deve ter no máximo seis (6) páginas.

Segue uma sugestão de estrutura do *relatório*:

- Uma introdução.
  - Uma introdução sobre a meta-heurística escolhida.
  - Uma descrição precisa e formal do problema escolhido (contendo a formulação matemática deste).
- Uma descrição do algoritmo incluindo:
  - A representação do problema.
  - As principais estruturas de dados
  - A heurística construtiva.
  - A vizinhança e a estratégia de escolha de vizinhos.
  - Os parâmetros do método (combinações testadas e valores usados nos experimentos).
  - O(s) critério(s) de parada.
- Uma tabela de resultados. Cada linha dessa tabela se refere a uma instância (i.e. o nome da instância é a chave primária da tabela) e a maioria das colunas traz médias de métricas geradas por dez execuções da *implementação* sobre cada instância (no caso de colunas referentes a *formulação matemática*, as métricas são provenientes de uma única execução). É necessário que a tabela contenha, pelo menos, as seguintes colunas:
  - O valor da solução inicial construtiva. O valor médio se a solução varia com a semente de aleatoriedade.
  - O valor médio da melhor solução encontrada pela sua implementação.
  - O desvio padrão das melhores soluções encontradas pela sua implementação.
  - O tempo médio de execução (em segundos).
  - O tempo de execução do GLPK (formulação matemática, em segundos). O limite de tempo deve ser no mínimo de 1h (pode ser obtido por meio do parâmetro `--tmlim 3600` do GLPK).
  - O valor da melhor solução encontrada pelo GLPK no tempo fornecido.

[2] O objetivo desta exigência, em conjunto com a exigência da apresentação das sementes de aleatoriedade, é garantir que o avaliador obterá os mesmos resultados descritos pelo aluno se executar o binário com a mesma semente de aleatóriedade e a mesma instância em sua máquina.

  - O desvio percentual $100\frac{UB-SM}{UB}$ do valor médio da solução encontrada pela sua implementação SM em relação ao valor da coluna *upper bound* UB para a instância específica (vide tabela 1, 2 ou 3).
- A análise dos resultados.
- As conclusões finais.
- Qualquer bibliografia utilizada para realização do trabalho.

A *implementação* deve ser executada no mínimo 10 vezes sobre cada instância. Cada execução deve ser realizada com uma semente de aleatoriedade diferente. Em C/C++, a semente de aleatoriedade é o valor passado ao procedimento `srand` antes de se começar a chamar `rand` (ou usando o construtor de `std::default_random_engine` da biblioteca padrão `<random>` em C++11)[3]. Caso seja utilizado o procedimento `srand`, este deve ser chamado somente uma vez no programa. Como cada execução com uma semente de aleatoriedade diferente pode ter resultados diferentes deve ser apresentada a média dos resultados das 10 execuções. Para k execuções, solicita-se usar as sementes de 1 a k.

4. Algumas referências externas sobre as meta-heurísticas

1. 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`)
2. 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`).
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. 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`)
5. Simulated Annealing: 'Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning, by D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, Operations research 37 (6), 865-892, 1989.' (`http://sci-hub.cc/10.1287/opre.37.6.865`)

[3]Mais informações em: `http://www.cplusplus.com/reference/random/`.

5. Problemas NP-completos permitidos

Knapsack Sharing	Temporal Knapsack	Knapsack Problem with Conflict Graph

1. **Knapsack Sharing**

Instância Uma instância é composta por uma única capacidade c e n items, cada item possui três características: um lucro l, um peso p e grupo g.

Objetivo Determinar quais itens serão colocados na mochila, de forma a *maximizar* o lucro do grupo com o *menor* lucro entre todos os grupos[4][5], enquanto respeitando a restrição de que a soma do peso de todos os itens (de todos os grupos) não pode ultrapassar a capacidade da mochila (i.e. só existe uma mochila com a capacidade c onde itens de todos os grupos serão colocados). Só existe uma cópia de cada item disponível, embora possam haver dois itens com os mesmos valores de lucro, peso e grupo.

Referência Hifi, M., Sadfi, S., & Sbihi, A. (2002). An efficient algorithm for the knapsack sharing problem. Computational Optimization and Applications, 23(1), 27-45.

2. **Temporal Knapsack**

Instância Uma instância é composta por uma única capacidade c, e n pedidos. Cada pedido consiste de lucro l, demanda d, tempo (segundo) de início s e de término t.

Objetivo Determinar quais pedidos serão aceitos, de forma a maximizar a soma dos lucros dos pedidos aceitos, enquanto respeitando a restrição de que, a cada segundo, a soma das demandas dos pedidos aceitos e ativos[6] não ultrapasse c.

Referência Bartlett, M., Frisch, A. M., Hamadi, Y., Miguel, I., Tarim, S. A., & Unsworth, C. (2005, May). The temporal knapsack problem and its solution. In International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming (pp. 34-48). Springer Berlin Heidelberg.

3. **Knapsack Problem with Conflict Graph**

Instância Uma instância é composta por uma única capacidade c, n itens, e m arestas.

Objetivo Determinar quais itens serão colocados na mochila, de forma a maximizar a soma do lucro dos itens, enquanto respeitando as restrições de que: a soma dos pesos dos itens selecionados não pode ultrapassar a capacidade da mochila; se existe uma aresta (i, j), onde i e j são dois items, a mochila não pode conter os itens i e j simultaneamente (i.e. a mochila pode ter nenhum dos dois itens, ou só o item i, ou só o item j, mas não os itens i e j).

Referência Pferschy, U., & Schauer, J. (2009). The Knapsack Problem with Conflict Graphs. J. Graph Algorithms Appl., 13(2), 233-249.

[4]Um exemplo é apresentado a seguir: caso existam dois grupos (A e B) e, em uma determinada solução parcial, o lucro de cada um destes grupos é, respectivamente, 22 e 78, então o valor da solução é 22. Considerando este exemplo, adicionar um item à solução que aumentasse o lucro do grupo B de 78 para 122 não alteraria o valor da função objetivo, enquanto aumentar o lucro dos itens no grupo A de 22 para qualquer valor entre 23 e 78 aumentaria o valor da função objetivo para este mesmo valor. Aumentar o lucro do grupo A para um valor *maior que* 78 aumentaria o valor da função objetivo para 78, já que o lucro do grupo B passaria a definir o valor da função objetivo.

[5]Uma dica para a implementação da formulação matemática: `https://en.wikibooks.org/wiki/GLPK/Modeling_tips#Extremum_terms`.

[6]Um pedido é considerado *ativo* em um determinado segundo, se o intervalo $[s, \ldots, t]$ do pedido contêm o segundo em questão.

5.1. Instâncias dos problemas

Os pacotes com as instâncias de cada problema estão disponíveis em: `www.inf.ufrgs.br/%7Ehbecker/ksp_instances.tar.gz` (Knapsack Sharing Problem), `www.inf.ufrgs.br/%7Ehbecker/tkp_instances.tar.gz` (Temporal Knapsack Problem), `www.inf.ufrgs.br/%7Ehbecker/kpcg_instances.tar.gz` (Knapsack Problem with Conflict Graph). Estas instâncias são as mesmas referenciadas nas tabelas apresentadas na última página.

O formato das instâncias de cada problema é descrito abaixo:

1. **Knapsack Sharing Problem:** A primeira linha do arquivo informa o número de items (n). A segunda linha do arquivo informa o número de grupos G. A terceira linha do arquivo informa a capacidade da mochila c. A quarta linha apresenta G números inteiros positivos, os quais somam n, e representam, cada um, a quantidade de items que faz parte do k-ésimo grupo (ex.: o primeiro número é 72, então os 72 primeiros items fazem parte do primeiro grupo, ...). As n linhas seguintes (da linha 5 até a $n + 4$), descrevem cada um dos n itens. Cada linha descrevendo um item contém, nesta ordem, o peso do item e o lucro do item (o grupo do item deve ser deduzido a partir da sua posição na lista de itens e dos valores apresentados na quarta linha).
2. **Temporal Knapsack Problem:** A primeira linha do arquivo informa o número de items (n). A segunda linha do arquivo informa a capacidade c disponível a cada segundo. As n linhas seguintes (da linha 3 até a $n + 2$) descrevem cada um dos n pedidos. Cada linha descrevendo um pedido contêm, nesta ordem, os seguintes valores (na forma de números inteiros separados por espaços): lucro, demanda, segundo de início, segundo de término. O arquivo não explicita qual é o primeiro e o último segundos para qual os pedidos fazem referência (essa informação deverá ser deduzida a partir da lista de pedidos).
3. **Knapsack Problem with Conflict Graph:** O arquivo é uma seção *data* de uma formulação matemática do GLPK/MathProg[7]. Os parâmetros n e c se referem ao número de itens e a capacidade, respectivamente. O set V se refere aos números de 0 a $n - 1$. Os parâmetros p e w se referem aos lucros (*profits*) e pesos (*weights*) dos itens. O set E se refere as arestas (*edges*) entre os itens (cada aresta é denotada por um par de índices de itens).

[7] A formulação exata não é disponibilizada para não dar uma vantagem aos grupos que escolherem esse problema. Caso a formulação elaborada pelo grupo não seja compatível com o formato do arquivo de instância, o grupo deve apresentar um programa/script que converta do formato oferecido para o formato usado pela sua formulação (como os grupos que escolheram outros problemas).

Tabela 1. Instâncias do Knapsack Sharing a serem consideradas nos testes computacionais.

Nome	#itens	#grupos	capacidade	*upper bound*
A02C	1000	2	12805	2042
A10	1000	10	13021	1464
A50	1000	50	12966	232
B50	2500	50	32416	669
C50	5000	50	63188	1357
E50	10000	50	126532	2783
F50	20000	50	254940	5632
HB150	22500	250	1125000	7307
HB200	40000	250	2000000	9743
HB250	62500	250	3125000	12179

Tabela 2. Instâncias do Temporal Knapsack a serem consideradas nos testes computacionais.

Nome	#itens	capacidade	*upper bound*
U2	1000	500	48947
U100	1000	501	27271
I5	3209	100	81212
I25	5875	100	105109
I72	3424	100	46629
I100	7709	100	98851
I90	13025	100	152989
HB5000	5000	1388	34551
HB10000	10000	2777	71759
HB15000	15000	4166	110833

Tabela 3. Instâncias do Knapsack Problem with Conflict Graph a serem consideradas nos testes computacionais.

Nome	#itens	#arestas	capacidade	*upper bound*
C10_BPPC_5_0_5.txt_0.1.dat	60	180	10000	12179
C1_BPPC_8_0_9.txt_0.1.dat	501	12550	1000	1040
C3_BPPC_1_0_2.txt_0.1.dat	120	720	450	600
R10_BPPC_1_0_6.txt_0.1.dat	120	720	1500	2007
R1_BPPC_2_0_5.txt_0.1.dat	250	3125	150	559
R3_BPPC_4_0_9.txt_0.1.dat	1000	50000	450	1763
test_1000_2000_r0.05-0.dat	1000	24975	2000	7310
HB10.dat	1024	102400	51200	49836
HB11.dat	2048	247808	102400	99702
HB12.dat	4096	589824	204800	199413