MO824 - Atividade GRASP

Ieremies Romero (217938)

Descrição do problema

Modelo matemático

A partir do problema da *Maximum Quadratic Binary Function* (MAX-QBF), proposto por [1], o problema de maximização da função $f : \mathbb{B}^{|x|} \to \mathbb{R}$, onde $|x|$ é a dimensão do problema, descrita como

$$f(x_1, \ldots, f(x_n)) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j} x_i x_j,$$

em que $a_{i,j} \in \mathbb{R}(i, j = 1, \ldots, n)$ são os coeficientes da função.

A este problema adicionamos a restrição

$$\sum_{i=1}^{n} w_i x_i \leq W,$$

onde w_i é dito o peso da variável x_i. Nosso objetivo nessa nova versão chamada *Maximum Knapsack Quadratic Binary Function* (MAX-KQBF) é maximizar a função, tal que a soma dos pesos das variáveis na solução não exceda W.

Nesse problema, nossas variáveis de decisão consistem em tornar ou não um certo x_i para valor 1 ou não. Isso será representado no código como inserir ou não na solução.

Durante esse relatório, nos referiremos a $c(x_i)$ como o custo que o elemento x_i incumbe a atual solução. Caso x_i já esteja na solução, ele é a diferença no valor da função objetivo da solução atual e sem ele. Caso x_i não pertença à solução, $c(x_i)$ é a diferença no valor da função objetivo da solução atual e com ele.

Metodologia

Para esse problema utilizaremos a meta-heurística **GRASP** (*greedy randomized adaptive search procedure*), proposta por [2] para problemas de minimização. Esta consiste em alternamos entre duas fases: uma fase construtiva e outra de busca local. Na primeira, utilizamos uma heurística gulosa aleatória para construir uma solução. Se esta não for viável, concertemo-na, tornando-a uma solução viável. Na segunda, realizamos uma busca local partindo desta solução. Por fim, repetimos esse processo um certo número de vezes, guardando a melhor solução encontrada.

É importante salientar que, como demonstrado em [**rezende19_grasp**], não é necessário fazer a etapa de "reparo" da solução se só incluirmos nela aqueles que não irão torná-la inviável. Tais variáveis são chamadas **candidatas** e, a cada iteração da etapa de construção, montamos (ou atualizamos) a chamada lista de candidatas **CL**.

Nesta etapa, a cada iteração analisamos cada elemento da lista CL e qual custo que sua inserção na solução irá causar. Em posse do maior e menor custos nesta lista, selecionamos aleatoriamente

elementos que estão suficientemente próximos do menor custo, tais elementos compõem a chamada **lista restrita de candidatas** (RCL). A definição de suficientemente fica a cargo do parâmetro α e é relativa ao intervalo de valores que obtivemos na análise da lista. Podemos repetir esse processo até que a lista de candidatos seja esgotada.

Já na etapa de busca local, analisamos as vizinhanças da nossa solução procurando por melhorias locais até não ser mais possível. Para cada problema, podem existir diversas definições de vizinhança e mais de um podem ser utilizadas nessa fase.

Heurística construtiva

Neste projeto, utilizamos como heurística padrão, um algoritmo muito similar ao proposto por [**rezende19_grasp**].

Iniciamos com uma lista de candidatos CL composta por todos os elementos podem ser adicionados a solução e que, se o feito, infringirão na restrição de peso. A cada interação, atualizamos CL com o mesmo critério, removendo aqueles que não podem mais serem adicionados.

Baseado no custo adicional que cada candidata incumbirá à solução quando inserida, determinamos quais os maiores ($maxCost$) e menores ($minCost$) custos e, baseado no parâmetro α, determinamos a lista de candidatos restritos de forma gulosa por

$$RCL = \{cand : c(cand) \leq minCost + \alpha\ (maxCost - minCost)\}.$$

Por fim, concluímos a interação da heurística realizando o passo aleatório: escolhemos um elemento de RCL aleatoriamente a ser inserido na nossa solução. Observe que o parâmetro α determina o espaço amostral que teremos para retirar nosso elemento aleatório. Caso este seja pequeno de mais, realizamos apenas uma heurística gulosa, caso seja grande de mais, temos um algoritmo extremamente aleatório. Nesse experimento trabalhamos com dois valores de α: $\alpha_1 = 0.05$ e $\alpha_2 = 0.17$.

Para a heurística construtiva, consideramos como critério de parada quando a nossa lista de candidatas CL se tornar vazia ou nenhuma inserção que possamos fazer irá melhorar nossa função objetivo.

Métodos construtivos alternativos

Algumas alterações nesse método construtivo padrão são apresentadas em [**rezende19_grasp**]. Dentre elas, aplicamos **random plus greedy** e **POP**.

Random plus greedy

Nessa variante para a heurística construtiva, ao invés de tomarmos decisões gulosas e depois aplicarmos a aleatoriedade, fazemos o inverso. A partir da lista CL, retiramos uma amostra aleatória de tamanho máximo p para compor nossa RCL. Dentro da RCL, escolhemos o elemento com menor custo.

Nesse experimento, inicialmente tratamos $p = 0.5|x|$, mas, como será observado na seção Resultados, outros valores também foram posteriormente testados.

POP

Proximate Optimality Principle (POP) baseia-se na ideia de que "boas soluções são encontradas perto de outras boas soluções". Assim, nessa variante, realizamos buscas locais em certos momentos da heurística construtiva. No nosso caso, sabemos que a heurística irá tomar no máximo $|CL|$ do CL inicial e, usando essa estimativa, realizamos uma busca local depois de 33% e 66% das iterações terem sido realizadas.

Busca local

Na etapa de busca local, como descrito anteriormente, partimos de uma solução viável e, analisando a(s) vizinhança(s) desta solução, tomamos "passos" em direção a melhorar nossa função objetivo. O desafio então jaz em decidir quem serão nossas vizinhanças já que qualidade da busca depende diretamente nelas.

As vizinhanças a serem utilizadas são 3, compostas pelas soluções viáveis que diferem da atual pela:

- inserção de um novo elemento da CL na solução.
- remoção de um elemento já presente solução.
- substituição de um elemento na solução por outro na CL.

É importante ressaltar que a cada passo, a lista CL é atualizada com os elementos que não infringirão a restrição de peso, garantindo que a busca local sempre irá permanecer com soluções viáveis.

Uma vez que as vizinhanças estão definidas, é importante determinar qual dos passos será tomado. Duas abordagens são estudadas nesse experimento: **best-improving** e **first-improving**. Na primeira, percorremos todos os vizinhos (definidos pelas vizinhanças acima) e tomamos o passo na direção do vizinho que melhor afeta nossa função objetivo (no caso nosso caso, o de maior contribuição). Em contrapartida, a segunda nos propõe a tomar o primeiro "bom vizinho", ou seja, o primeiro vizinho encontrado que melhora a nossa solução.

Para a busca local, iteramos até que não haja nenhum vizinho que melhore nossa solução.

Resultados

Realizamos os experimentos das metodologias apresentadas acima em uma máquina com processador AMD Ryzen 1800x (16MB cache, 3.6GHz max boost 4.0GHz, 8 cores e 16 threads), 16GB DDR4-SDRAM, 240 GB SSD, Nvídia GeForce GTX 1060 com 6GB de vRam. A máquina estava configurada com o sistema operacional Windows 19 (versão 21H1).

Utilizamos extensivamente o framework em java disponibilizado pelo professor, mantendo nossas alterações mínimas.

As instâncias utilizadas para teste também foram disponibilizadas pelo professor da disciplina e diferem pela quantidade de variáveis no $KQBF$, ou seja, $|x|$. Para cada instância, realizamos 1000 iterações da meta-heurística e estipulamos um tempo limite de 30 minutos.

Partimos da configuração padrão como $\alpha = 0.05$, heurística construtiva padrão e busca local com *first-improving*. Quando não explicitado, os parâmetros se mantém igual à configuração padrão.

Inicialmente, estudamos 5 configurações, alterando apenas um aspecto por vez:

Configuração 1 configuração padrão.

Configuração 2 $\alpha = 0.17$.

Configuração 3 busca local utilizando o *best-improving*.

Configuração 4 heurística *random-plus-greedy*.

Configuração 5 heurística *POP*.

Table 1: Resultados obtidos para cada uma das configurações iniciais, em cada uma das instâncias. O tempo é mostrado em segundos.

	padrão		$\alpha_2 = 0.17$		best		random plus		pop	
$\|x\|$	Custo	Tempo	Custo	Tempo	Custo	Tempo	Custo	Tempo	Custo	Tempo
20	104	0.161	120	0.175	104	0.168	120	0.137	104	0.209
40	284	0.448	316	0.432	284	0.488	308	0.301	259	0.436
60	482	1.321	473	1.569	482	1.586	491	0.938	475	1.839
80	781	3.102	769	3.346	781	3.892	774	1.82	757	3.471
100	1221	5.788	1164	6.441	1221	6.343	1191	3.307	1192	6.698
200	3748	51.278	3818	56.356	3837	62.164	3733	27.095	3586	55.861
400	10193	636.808	10125	936.626	10132	1137.632	10136	266.525	9851	654.087

A tabela 1 demonstra os valores absolutos dos resultados.

Porém, uma visão mais interessante seria comprar em termos relativos, as melhorias. A Figura 1 atende justamente a isso. Nela, podemos observar que nenhuma das variações parece resultar em diferenças consideráveis no resultado. Acreditamos que isso se dá ao tamanho das instâncias não serem tão grandes comparados à quantidade de iterações realizadas, permitindo que eventualmente cheguem perto da mesma solução.

A grande diferença se mostra comparando o tempo de execução. Todas as alterações se mostraram pior no tempo que a configuração original com exceção da *random-plus-greedy*, cuja melhora no tempo significativa não veio a custo de piora na solução. Ainda sobre essa métrica, é importante salientar que a instância de 400 variáveis na configuração 5 foi a única a atingir o tempo limite de 30 minutos, o que faria da última coluna do seu gráfico ainda maior.

Em mãos desses resultados, achamos interessante aprofundarmos na heurística *random-plus-greedy*, experimentando com diferentes valores para p, como demonstrado na Figura 2. Interessantemente, observamos que o valor de p relativo ao tamanho da instância não aparenta fazer tanta diferença nos valores testados, sendo consistentemente melhor que a configuração padrão em tempo e mantendo o nível de solução. Mais uma vez, acreditamos que isso se dá ao alto número de interações comparado ao baixo número de variáveis.

References

[1] Hao Kochenberger, Lewis Glover, and Wang Lü. "Wang, 2014 Kochenberger G". In: *Hao JK, Glover F., Lewis M., Lü Z., Wang H., Wang Y., The unconstrained binary quadratic programming problem: a survey, J. Comb. Optim* 28.1 (2014), pp. 58–81.

[2] Mauricio GC Resende and Celso C Ribeiro. "Greedy randomized adaptive search procedures: advances and extensions". In: (2019), pp. 169–220.

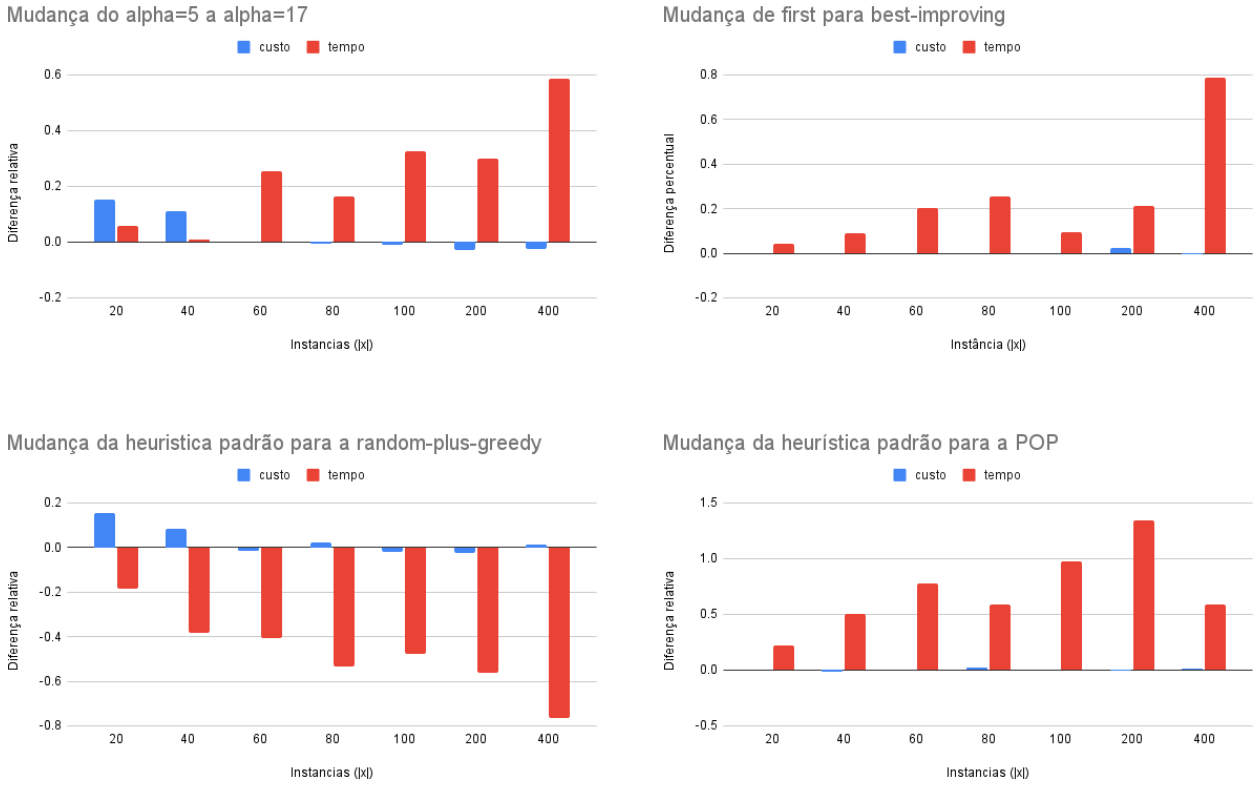

Figure 1: Gráficos relativos comparando as configurações [2-5] com a configuração padrão. Em azul as diferenças de custo (quanto maior, melhor) e em vermelho as diferenças de tempo em segundos (quanto menor, melhor).

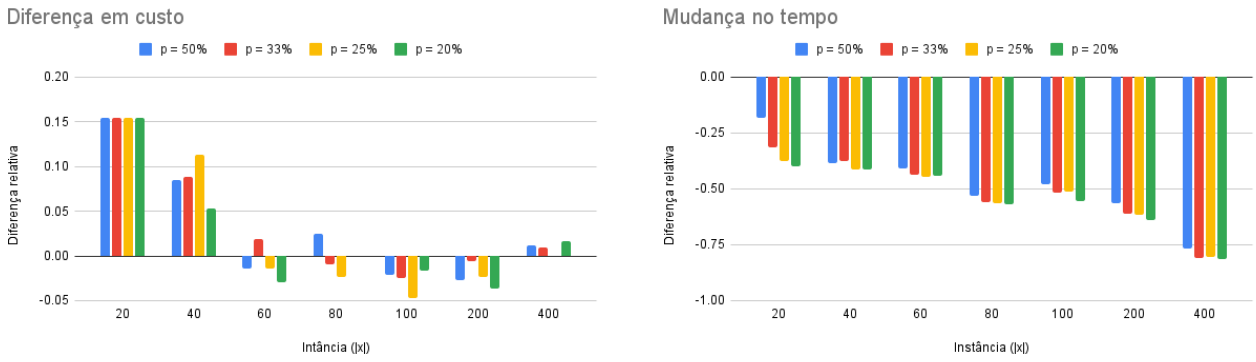

Figure 2: Gráficos relativos comparando as configurações de diferentes valores de p para a heurística random-plus-greedy com a configuração padrão. A direita as diferenças de custo, a esquerda as diferenças de tempo.