Submetido para MO824

Atividade 4 - Grupo 5 - MO824

Victor Ferreira Ferrari - RA 187890
Gabriel Oliveira dos Santos - RA 197460
Vitor Satoru Machi Matsumine - RA 264962

Resumo. Este trabalho apresenta soluções heurísticas para o problema MAX-QBFPT baseadas na metaheurística GRASP. Foram explorados os métodos construtivos Funções de Viés e *Random plus Greedy*, além das estratégias de busca local *first-improving* e *best-improving*. Nossos experimentos demonstraram a eficiência da abordagem *first-improving* comparada a *best-improving*, e a eficácia do uso de função de viés linear. Os resultados obtidos são bons, atingindo ou ultrapassando os valores de referência utilizados, mas um único método não pode ser considerado ideal, ou melhor em todas as instâncias.

Palavras-chave. GRASP, Otimização, QBFPT, QBF, metaheurística.

1. Introdução

Este trabalho consiste na apresentação de soluções heurísticas, baseadas na metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*), para o problema MAX-QBF com triplas proibidas, proposto na Atividade 4 de MO824 2S-2020. Para isso, foram realizados experimentos testando dois dos métodos construtivos apresentados por Resende e Ribeiro, em "Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications" [1].

O Problema MAX-QBF com Triplas Proibidas (MAX-QBFPT) é uma variação do Problema MAX-QBF ("Maximum quadractic binary function"), em que se objetiva maximizar uma função binária quadrática, de modo que x_i, x_j e x_k não assumam o valor 1 simultaneamente caso $(i,\ j,\ k)$ forme uma tripla proibida. O MAX-QBF é comprovadamente NP-Difícil [2] quando todos os coeficientes a_{ij} são não negativos.

2. Modelo Matemático

Uma QBF é uma função binária quadrática ($f : \mathbb{B}^n \to \mathbb{R}$) que pode ser expressa como uma soma de termos quadráticos, com variáveis binárias. Assim, ao se formular um modelo matemático para o problema, as variáveis binárias do problema são imediatamente traduzidas para variáveis do modelo.

Seja x_i variáveis binárias de uma QBF $f(x_1, \ldots, x_n)$, para $i \in \{1 \ldots n\}$ e $\mathcal{T} = \{(i, j, k) \in \mathbb{N} : 1 \leq i < j < k \leq n\}$ o conjunto de todas as triplas ordenadas,

sem repetição, dos naturais de 1 a n. Então o problema MAX-QBFPT dado o conjunto de triplas proibidas $T \subseteq \mathcal{T}$, consiste na maximização de $f(x_1, \ldots, x_n)$, de modo que x_i, x_j e x_k não sejam todos iguais a 1, para todo $(i, j, k) \in T$. Assim, o MAX-QBFPT pode ser formulado da seguinte forma:

$$\max \quad Z = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot x_i \cdot x_j$$

$$\begin{aligned} &s.a. \\ &\quad x_i + x_j + x_k \leq 2 && \forall (i, j, k) \in T \\ &\quad x_i \in \{0, 1\} && \forall i = 1, ..., n \end{aligned}$$

Onde $a_{ij} \in \mathbb{R}$ $\forall i, j = 1, .., n$ são parâmetros do problema. A função objetivo é a QBF, e as restrições são relacionadas às triplas proibidas e ao domínio, respectivamente. O conjunto T pode ser qualquer subconjunto $\mathcal{T}$ de triplas ordenadas, sem repetições.

2.1. Conjunto de Triplas Proibidas

Especificamente para esta atividade, as triplas são geradas a partir de duas funções $g, h : [1, n] \rightarrow [1, n]$, que são definidas do seguinte modo:

$$l_1(u) = 1 + ((\beta_1 \cdot (u - 1) + \beta_2) \; mod \; n)$$

$$g(u) = \begin{cases} l_1(u), & \text{se } l_1(u) \neq u \\ 1 + (l_1(u) \; mod \; n), & \text{caso contrário} \end{cases}$$

$$l_2(u) = 1 + ((\pi_1 \cdot (u - 1) + \pi_2) \; mod \; n)$$

$$h(u) = \begin{cases} l_2(u), & \text{se } l_2(u) \neq u \text{ e } l_2(u) \neq g(u) \\ 1 + (l_2(u) \; mod \; n), & \text{se } (1 + (l_2(u) \; mod \; n)) \neq u \text{ e} \\ & \quad (1 + (l_2(u) \; mod \; n)) \neq g(u) \\ 1 + ((l_2(u) + 1) \; mod \; n), & \text{caso contrário} \end{cases}$$

Onde os valores de β_1, β_2, π_1 e π_2 são respectivamente 131, 1031, 193 e 1093.

A partir dessas funções definimos o conjunto de triplas proibidas como $T = \{(i, j, k) \in \mathcal{T} : \forall u \in [1, n], (i, j, k) = sort(\{u, g(u), h(u)\})\}$.

3. Metodologia

O GRASP é uma meta heurística iterativa proposta em 1995 por Feo e Resende que surge da ideia de criar um método que utiliza busca local com múltiplos pontos de início. Em termos básicos, uma iteração do método consiste na construção de uma solução gulosa aleatorizada, passando por uma possível fase de viabilização, e terminando com uma busca local tomando como base a solução construída. Esta

seção detalhará como cada passo do método foi feito para o problema MAX-QBFPT, e descreverá os métodos alternativos de construção e busca local testados.

Os critérios de parada utilizados foram número de iterações e tempo máximo.

3.1. Heurística Construtiva

A construção das soluções gulosas aleatorizadas, em cada iteração do GRASP, é feita segundo o pseudocódigo descrito em [1]. Uma lista de candidatos (C) é iniciada com todas variáveis x_i da QBF. A partir dessa lista, cada candidato pode ser inserido ou não na solução, e essa decisão é feita a partir de uma lista **restrita** de candidatos (RCL).

A cada iteração da heurística, a lista de candidatos é **atualizada** a partir da solução atual, removendo da lista os elementos que completariam triplas proibidas, ou seja, filtrando a lista de candidatos para manter apenas variáveis que podem compor soluções **factíveis**. Em seguida, os candidatos com menor custo (c_m) e maior custo (c_M) são encontrados e salvos. Assim, a RCL é dada por $\{e \in C | c_e \leq c_m + \alpha(c_M - c_m)\}$, onde α é o parâmetro que controla a gulosidade e aleatoriedade, como indicado em [1].

A partir da RCL, um elemento é escolhido de forma aleatória, e é removido da lista de candidatos e inserido na solução, como detalhado na Seção 3.1.1. A Construção é finalizada se a lista de candidatos é vazia ou se o custo da solução proposta na iteração atual não for melhor que o custo da solução da iteração anterior.

Nessa atividade escolhemos implementar duas heurísticas construtivas: Funções de Viés (*Bias Functions*) e Random plus Greedy.

3.1.1. Funções de Viés

Nessa abordagem, a escolha aleatória de um candidato da RCL a ser inserido na solução em construção é feita a partir de uma **função de viés**. Nela, são atribuídas probabilidades para cada elemento com base em uma função $\texttt{bias}(r)$ e no valor de inserção do elemento na solução.

Dado um candidato σ, podemos definir um *rank* $r(\sigma)$ para o elemento dentro do conjunto C. Assim, a probabilidade de σ é dada pela Equação (3.1).

$$\pi(\sigma) = \frac{\texttt{bias}(r(\sigma))}{\sum_{\sigma' \in C} \texttt{bias}(r(\sigma'))} \tag{3.1}$$

Esse método não elimina diversidade, pois a escolha ainda é feita aleatoriamente, mas fornece um viés para essa escolha, tentando aumentar a qualidade média das soluções.

A escolha padrão do GRASP ainda pode ser feita, definindo a função `bias` constante igual a 1, independente do *rank*, e assim as probabilidades são iguais para todos os elementos da RCL. Porém, outras combinações podem ser testadas, como funções lineares, logarítmicas, exponenciais e polinomiais [1]. Após diversos testes com todas as funções citadas, escolhemos focar em uma função de viés **linear**.

3.1.2. Random plus Greedy

Na construção da RCL padrão, o parâmetro α desempenha papel importante, controlando o *trade off* entre aleatoriedade e gulosidade. Esse parâmetro define o limite de qualidade dos elementos da RCL na fase de construção. Valores menores de α indicam uma construção mais gulosa, e valores maiores indicam construção mais aleatória. Atingir um equilíbrio entre qualidade e diversidade é essencial para um bom método.

No método *Random plus Greedy* esse parâmetro α é substituído pelo parâmetro $p \in \mathbb{N}$, de modo que os primeiros p elementos da solução são escolhidos de modo completamente aleatório. Em seguida, os elementos finais são escolhidos de modo completamente guloso. Nessa abordagem, o controle do *trade off* entre aleatoriedade e gulosidade é feito pelo ajuste de p, onde valores maiores resultam em soluções mais aleatórias, e valores menores resultam em soluções mais gulosas.

3.2. Busca Local

A etapa de busca local consiste na exploração da vizinhança da solução S construída anteriormente. Para o problema MAX-QBFPT, uma vizinhança imediata consiste nas soluções obtidas ao **inserir** ou **remover** um elemento em S, ou **trocar** um elemento de S por outro. Essas operações são chamadas de *movimentos de vizinhança*. A lista de candidatos é atualizada a cada iteração, para conter apenas elementos factíveis, que não violem as triplas proibidas.

Dados os movimentos, a vizinhança é explorada até que um ótimo local seja encontrado, sendo este retornado para atualização da solução titular. A exploração dessa vizinhança pode ser feita de dois modos: *first-improving* e *best-improving*.

A estratégia *best-improving* trata de avaliar todos os movimentos possíveis a partir da solução atual, e escolhe aquele que fornece a maior melhoria. Em outras palavras, todas as inserções, remoções e trocas são avaliadas, e o melhor movimento é escolhido.

Para a estratégia *first-improving*, foi considerada a melhor primeira escolha dentre as três possibilidades de modificação (inserção, remoção e troca) que causa uma melhor maximização. Para tal, tomamos o primeiro elemento da lista de candidatos cuja inserção causaria uma melhora no custo. Caso a remoção de algum candidato apresente um custo melhor que a primeira escolha da inserção, substituímos pela primeira remoção encontrada que supera este custo; um processo similar é feito para a troca, considerando a melhor primeira escolha entre a inserção e a remoção.

4. Implementação e Avaliação

Assim como indicado na atividade, foram utilizados dois métodos de busca (*first-improving* e *best-improving*), dois valores para o parâmetro $\alpha \in [0, 1]$, dois métodos de construção alternativos: funções de viés e *random-plus-greedy* (RPG). Como descrito na seção 3.1.1, a função de viés linear foi escolhida por gerar os melhores resultados dentre as testadas. Os métodos de construção alternativos são aplicados

em momentos diferentes da heurística, como visto em 3.1.1 e 3.1.2, então ambos podem ser aplicados simultaneamente. As configurações testadas foram:

1. PADRÃO: GRASP com parâmetro $\alpha_1 = 0.25$, *best-improving* e heurística construtiva padrão.
2. PADRÃO+HC1: GRASP padrão com método construtivo função de viés linear.
3. PADRÃO+HC1+ALPHA: GRASP padrão com método construtivo função de viés linear e $\alpha_2 = 0.7$.
4. PADRÃO+HC2+HC1: GRASP padrão com método construtivo *Random plus Greedy* ($p = 2$) e função de viés linear.
5. PADRÃO+HC2+HC1+FIRST: GRASP padrão com método construtivo *Random plus Greedy* ($p = 2$), função de viés linear e *first-improving*.

A implementação da solução foi feita em *Java*, com base no *framework* fornecido. Alteramos o código original para incluir o critério de parada por limite de tempo de execução, para parar a construção de uma solução gulosa caso a lista restrita de candidatos (RCL) esteja vazia, seguindo o pseudo código indicado em [1], e para implementar as modificações citadas na Seção 3.

As instâncias utilizadas em nossos experimentos foram fornecidas previamente no pacote de atividades. Tais instâncias foram criadas originalmente para o problema MAX-QBF, porém como também foram geradas o conjunto de triplas proibidas, foi possível reaproveitá-las. Algumas instâncias possuem solução ótima conhecida, outras possuem intervalos de referência, que podem ser usados para comparação.

O *hardware* usado para testes possui uma CPU "Intel(R) Core(TM) i7-9750H (6C/12T)", com 2.60 GHz de frequência do *clock*. O computador também possui 16 GB de RAM, operando com 2660 MHz. O sistema operacional utilizado foi o Windows 10 (64 bits). O método foi executado com limite de 1800 segundos (30 minutos), limite de 1000 iterações, e sem limite de memória.

5. Resultados Obtidos e Análise

Nas Tabelas 1, 2 e 3 são apresentados os resultados obtidos. O tempo é medido em segundos e o número de iterações realizadas é acompanhado pela iteração na qual o valor máximo foi obtido (MAX). A corretude da geração de triplas foi testada separadamente, e os testes foram bem-sucedidos.

Os resultados mostram que não houve um método ideal no qual foi possível obter as melhores soluções para todas as instâncias, ou seja, nenhum dos métodos alcançou a solução de referência para todas as instâncias. Nesse quesito, a melhor configuração testada é a 3, com viés linear e $\alpha = 0.7$ (Tabela 2), atingindo ou ultrapassando os limitantes inferiores de referência em todas as instâncias exceto a de tamanho 80, e necessitando de poucas iterações para atingir esse resultado.

Isso faz sentido pois o a função de viés pode ter mais impacto com uma RCL maior, comparando com a mesma configuração com $\alpha = 0.25$ (Tabela 1). Por outro lado, também é a configuração mais lenta, sendo a única que não conseguiu atingir 1000 iterações no tempo alocado com a instância de tamanho 200. Isso se deve ao maior número de candidatos escolhidos para a RCL em cada iteração da heurística construtiva. Porém, como necessita de menos iterações para encontrar uma boa solução, se consolida como uma boa configuração.

Em geral, os resultados obtidos foram bons, ultrapassando a solução de referência das duas maiores instâncias em todos os métodos. Se filtrarmos os melhores resultados de cada método, todas as soluções ótimas conhecidas foram alcançadas, e todas as soluções de referência conhecidas foram alcançadas ou ultrapassadas.

Com um valor mais guloso de α, o viés linear não tem um grande impacto positivo nos resultados. Porém, como citado anteriormente, ampliar a RCL tem um ótimo efeito. Em geral, resultados iguais ou melhores são obtidos, em menos iterações. Por esse motivo, foi escolhido manter essa função para os experimentos posteriores. Outros testes foram feitos com a construção HC2 sem viés, mas os resultados obtidos foram piores.

A combinação do *Random plus Greedy* (RPG) com a função de viés linear na heurística construtiva (configurações 4 e 5), apresentou também ótimos resultados para quase todas as instâncias, com os melhores valores para as instâncias 200 e 400 na configuração 5. A configuração 4 também obteve o mesmo resultado da configuração 3 na instância 100 (ambos os melhores com um valor de 1263), porém com um tempo de execução significativamente menor. Essas combinações também foram as únicas que alcançaram a referência na instância 80. Por outro lado, a instância 40 foi provavelmente prejudicada pelas fases gulosas da construção, já que essas configurações foram as únicas que não alcançaram a solução ótima.

É notável a diferença em tempo por iteração da estratégia *first-improving* em comparação com a *best-improving*. Ela pode ser melhor identificada na instância de tamanho 400, na qual o único método que terminou por limite de iterações em vez de por tempo foi o que utilizava busca local *first-improving*. Comparando as configurações 4 e 5 (Tabelas 2 e 3), podemos ver que a configuração 4 executou 139 para a instância de tamanho 400, quanto que a configuração 5 atingiu o limite de iterações com 1065s. Além disso, para instâncias maiores a estratégia de busca *first-improving* retornou melhores resultados, provando sua utilidade e relevância frente às anteriores. Sendo assim, esta estratégia se mostra muito mais interessante em instâncias maiores, obtendo resultados semelhantes nas instâncias menores.

Referências

[1] M. G. C. Resende and C. C. Ribeiro, "Greedy randomized adaptive search procedures: Advances, hybridizations, and applications," in *Handbook of Metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), vol. 146 of *International Series in Operations Research & Management Science*, ch. 10, pp. 283–318, Springer Science+Business Media, 2010.

[2] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang, "The unconstrained binary quadratic programming problem: a survey," *Journal of Combinatorial Optimization*, vol. 28, no. 1, pp. 58–81, 2014.

Tabela 1: Resultados das configurações 1 e 2.

Instância	Referência	PADRÃO			PADRÃO+HC1		
		Valor	Iterações (MAX)	Tempo (s)	Valor	Iterações (MAX)	Tempo (s)
20	125	125	1000 (32)	0.757	125	1000 (11)	0.343
40	366	366	1000 (179)	3.236	366	1000 (497)	1.184
60	[508, 576]	508	1000 (180)	17.111	508	1000 (64)	5.849
80	[843, 1000]	832	1000 (601)	44.996	838	1000 (522)	15.866
100	[1263, 1539]	1227	1000 (565)	70.389	1217	1000 (995)	35.291
200	[3813, 5826]	3891	1000 (779)	393.138	3938	1000 (295)	424.488
400	[9645, 16625]	10892	271 (100)	1804.112	10747	173 (43)	1808.129

Tabela 2: Resultados das configurações 3 e 4.

Instância	Referência	PADRÃO+HC1+ALPHA			PADRÃO+HC2+HC1		
		Valor	Iterações (MAX)	Tempo (s)	Valor	Iterações (MAX)	Tempo (s)
20	125	125	1000 (23)	0.365	125	1000 (618)	0.36
40	366	366	1000 (75)	2.906	335	1000 (51)	1.879
60	[508, 576]	508	1000 (48)	12.825	508	1000 (469)	6.66
80	[843, 1000]	830	1000 (33)	41.439	843	1000 (996)	18.605
100	[1263, 1539]	1263	1000 (51)	107.794	1263	1000 (735)	42.623
200	[3813, 5826]	3939	905 (60)	1800.841	3937	1000 (97)	633.385
400	[9645, 16625]	10794	38 (14)	1807.672	10958	139 (19)	1822.587

Tabela 3: Resultados da configuração 5.

Instância	Referência	PADRÃO+HC2+HC1+FIRST		
		Valor	Iterações (MAX)	Tempo (s)
20	125	125	1000 (618)	0.262
40	366	335	1000 (51)	0.982
60	[508, 576]	508	1000 (469)	2.979
80	[843, 1000]	843	1000 (996)	7.837
100	[1263, 1539]	1247	1000 (475)	15.927
200	[3813, 5826]	3945	1000 (97)	130.501
400	[9645, 16625]	11088	1000 (398)	1064.824