Submetido para MO824

Atividade 6 - Grupo 5 - MO824

Rachel Vanucchi Saraiva - RA 185961
Sinara Caonetto Pamplona - RA 187101
Victor Ferreira Ferrari - RA 187890

Resumo. Este trabalho apresenta soluções heurísticas para o problema MAX-QBFPT baseadas no paradigma de Algoritmos Genéticos. Foram exploradas duas estratégias evolutivas além da padrão: *Steady-state* e manutenção de diversidade. Os métodos implementados foram avaliados através de 7 experimentos computacionais de 7 instâncias cada. De acordo com os resultados obtidos, as estratégias que proporcionaram resultados melhores foram aquelas que utilizaram *Steady-state* com manutenção de diversidade.

Palavras-chave. Algoritmos Genéticos, Otimização, QBFPT, QBF, metaheurística.

1. Introdução

Este trabalho consiste na apresentação de soluções heurísticas, baseadas no paradigma (ou metaheurística) Algoritmos Genéticos, para o problema MAX-QBF com triplas proibidas, proposto na Atividade 6 de MO824 2S-2020. Para isso, foram realizados experimentos testando duas das estratégias evolutivas apresentadas por Colin R. Reeves, em "Genetic Algorithms" [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.

1.1. 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 utilizando variáveis binárias. Sejam $a_{ij} \in \mathbb{R}$ $(i, j = 1, \ldots, n)$ os coeficientes de f, no problema em que estamos interessados, uma QBF é dada por:

$$f(x_1, ..., x_n) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot x_i \cdot x_j \tag{1.1}$$

Sejam x_i as 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. Dado um conjunto de triplas proibidas $T \subseteq \mathcal{T}$, o problema MAX-QBFPT 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 cada $(i, j, k) \in T$. Com base nisso, o MAX-QBFPT pode ser formulado como segue:

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

$$\text{s.a} \quad x_i + x_j + x_k \leq 2, \qquad \forall (i, j, k) \in T \quad (1.2)$$

$$x_i \in \mathbb{B}. \qquad \forall i \in \{1, ..., n\} \quad (1.3)$$

As restrições (1.2) correspondem às restrições das triplas proibidas, isto é, impedem que toda solução contenha um subconjunto qualquer de triplas ordenadas sem repetições e proibidas. As restrições (1.3) impõem o domínio das variáveis de decisão como sendo binárias. A função objetivo maximiza o valor da QBF.

1.2. 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 $T = \{(i, j, k) \in \mathcal{T} : \forall u \in [1, n], (i, j, k) = sort(\{u, g(u), h(u)\})\}$.

2. Metodologia

A metodologia a ser utilizada é a de Algoritmo Genético introduzida por John H. Holland em 1975 [2], metaheurística inspirada pelo conceito biológico de cruzamento seletivo e a forma como os genes dos filhos são combinações dos genes dos pais. Algoritmos Evolutivos, de maneira geral, utilizam conceitos chaves da biologia evolutiva,

como hereditariedade, seleção natural, mutação e reprodução/recombinação, sendo estas utilizadas nos experimentos desta atividade.

Como panorama geral da sequência dos passos utilizados em um algoritmo genético, veja abaixo o pseudocódigo do *framework* disponibilizado na disciplina.

Algoritmo 1: Algoritmo Genético

```
1:  populacao = inicializarPopulacao();
2:  melhorCromossomo = melhorCromossomo(populacao);
3:  melhorSolucao = decodificar(melhorCromossomo);
4:  for geracao <= maxGeracoes do
5:     pais = selecionarPais(populacao);
6:     descendentes = recombinar(pais);
7:     mutantes = mutacionar(descendentes);
8:     novaPopulacao = selecionarPopulacao(populacao, mutantes);
9:     populacao = novaPopulacao;
10:    melhorCromossomo = melhorCromossomo(populacao);
11:    if fitness(melhorCromossomo) > custo(melhorSolucao): then
12:       melhorSolucao = decodificar(melhorCromossomo);
13:    end if
14:    if tempo-total > tempo-permitido then
15:       pare;
16:    end if
17: end for
18: retorne melhorSolucao;
```

Em um algoritmo genético, as soluções são codificadas como *cromossomos*, que podem usar diversas estruturas de dados, comumente *strings* binárias. Cada elemento dessa *string* é um *gene*. Para o MAX-QBFPT, cada cromossomo é representado por um vetor de inteiros (são usados apenas 0 e 1), representando *strings* binárias em que o i-ésimo gene representa o valor de x_i na solução correspondente.

A cada iteração é mantido um conjunto de cromossomos, chamado de população, em um vetor. O tamanho da população é parametrizada. A população inicial é gerada de forma aleatória, através da adição de cromossomos retornados por uma função pseudoaleatória que atribui valores binários para cada gene. As próximas populações (gerações) são geradas a partir da recombinação (*crossover*) e mutação da população atual.

Cada cromossomo possui um valor de *fitness*, que serve como medida da qualidade da solução correspondente. Para este problema foi utilizado o próprio valor da solução como *fitness*. Esse valor deve ser maximizado durante a execução, e a melhor solução encontrada é retornada.

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

2.1. Viabilização

Na geração aleatória inicial de soluções, há uma grande possibilidade de violação de triplas proibidas, ou seja, soluções **inviáveis** para o problema. Inicialmente, foi testado um sistema de penalização de soluções inviáveis, mas em instâncias maiores nenhuma instância viável foi gerada durante a execução inteira.

Por isso, um processo de **viabilização** foi feito após a geração de cada cromossomo. Cada tripla é guardada em uma lista indexada por seu primeiro elemento. Assim, as triplas localizadas no índice de cada gene ativo são verificadas, em ordem. Caso uma violação seja detectada, um elemento **aleatório** da tripla é removido da solução, viabilizando o cromossomo em relação àquela tripla e não acarretando na violação de outras.

Uma análise mais complexa pode ser feita para a escolha de remoção, mas realizá-la aleatoriamente ajuda a preservar diversidade e é mais eficiente. O mesmo processo é aplicado nos cromossomos de cada população gerada, entre a mutação e a substituição de gerações, e deste modo a avaliação dos cromossomos é sempre feita com apenas soluções viáveis.

2.2. Seleção e Reprodução

Para o problema, foi usado um método *crossover-AND-mutation*, no qual ambas estratégias são usadas em cada geração. O *crossover* é feito para cada par de pais consecutivos em um conjunto previamente selecionado, com o mesmo tamanho da população inicial.

A técnica de *crossover* utilizada no método padrão foi a de *two-point crossover* (2X), na qual duas posições (*crosspoints*) dos cromossomos são escolhidas aleatoriamente. Os genes entre esses dois pontos de cruzamento são trocados entre os cromossomos, gerando dois "filhos", com características de ambos os pais e possivelmente diferentes entre si.

A escolha dos pais é feita pela estratégia de *seleção por torneio*. Dois cromossomos são escolhidos aleatoriamente e seus valores de *fitness* são comparados. O melhor dos dois cromossomos é escolhido para ser pai. Essa escolha aleatória é feita **com reposição** até que o conjunto de pais esteja com o tamanho da população, ou seja, com certeza nesse conjunto haverá repetição de pais.

2.3. Mutação

No processo de mutação, genes de um cromossomo são escolhidos aleatoriamente e têm seu valor alterado. Como os cromossomos utilizados para o MAX-QBFPT são vetores representando *strings* binárias, essa alteração é simplesmente inverter o valor de um gene.

A escolha dos genes que sofrem mutação é controlada por um parâmetro, a taxa de mutação, que indica a probabilidade de cada gene ser escolhido.

2.4. Substituição de Gerações

No método padrão, a estratégia utilizada para seleção da nova população é a de *elitismo*, na qual a nova população consiste a princípio de todos os filhos gerados pela atual, e o pior cromossomo dos filhos é substituído pelo melhor indivíduo da população atual. Essa estratégia garante que há ao menos uma solução boa da geração anterior preservada.

2.4.1. Steady-State

Uma estratégia alternativa também implementada é a chamada *steady-state*, na qual os melhores cromossomos dentre pais e filhos são escolhidos para compor a nova população, de forma a manter o mesmo tamanho da população anterior. Isso é chamado de *steady-state* $\lambda + \mu$ [1].

Essa técnica garante que não há perda de soluções boas de uma geração para a próxima, porém pode levar a convergência prematura, e baixa diversidade, graças a uma pressão seletiva alta.

2.4.2. Manutenção de Diversidade

Um fator importante para o bom funcionamento de um algoritmo genético é a diversidade da população, para que o algoritmo continue gerando novas soluções e explorando mais o espaço de busca. Como mencionado na seção anterior, o método *steady-state* possui uma pressão seletiva alta, então tende a perder diversidade muito fácil. Por isso, técnicas específicas para manutenção de diversidade são implementadas.

A técnica implementada neste projeto para a manutenção dessa diversidade consiste na escolha mais cautelosa dos *crosspoints* a serem usados no processo de *crossover*, de forma a evitar a geração de filhos idênticos. Utilizamos a operação XOR entre os pais para identificar os genes que eles diferem entre si como possíveis *crosspoints*. Com essa estratégia, os *crosspoints* são escolhidos aleatoriamente não a partir de todas as posições possíveis de genes, e sim apenas entre aquelas que podem proporcionar maior diversidade no cruzamento. Isso garante que os dois filhos gerados a partir desses pais também terão genes distintos nessa posição.

3. Implementação e Avaliação

Assim como indicado na atividade, foram utilizados dois tamanhos de população, duas taxas de mutação, e duas estratégias evolutivas alternativas: *steady-state* como método de substituição de população, e manutenção de diversidade para evitar convergência prematura no *steady-state*.

As configurações utilizadas nos experimentos computacionais foram:

1. PADRÃO: algoritmo genético com população de tamanho $P_1 = 100$, taxa de mutação $M_1 = 0.01$, elitismo sem controle de diversidade.

2. PADRÃO+STSTATE: algoritmo genético PADRÃO mas com *steady-state* como estratégia de substituição de população.
3. PADRÃO+STSTATE+DIV: algoritmo genético PADRÃO mas com *steady-state* como estratégia de substituição de população e manutenção de diversidade.
4. PADRÃO+STSTATE+DIV+POP: algoritmo genético com *steady-state* e manutenção de diversidade, mas com população de tamanho $P_2 = 50$.
5. PADRÃO+STSTATE+DIV+MUT: algoritmo genético com *steady-state* e manutenção de diversidade, mas com taxa de mutação $M_2 = 1/n$, onde n é o tamanho da instância.
6. PADRÃO+POP+MUT: algoritmo genético PADRÃO, mas com população de tamanho $P_2 = 50$ e taxa de mutação $M_2 = 1/n$, onde n é o tamanho da instância.
7. PADRÃO+STSTATE+DIV+POP+MUT: algoritmo genético com *steady-state* e manutenção de diversidade, mas com população de tamanho $P_2 = 50$ e taxa de mutação $M_2 = 1/n$, onde n é o tamanho da instância.

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 e para implementar as modificações citadas na Seção 2.

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. A configuração e os limitantes inferiores e superiores do valor ótimo de cada instância são apresentados na Tabela 1.

Tabela 1: Instâncias utilizadas nos experimentos

i	Instância	n	MAX-QBF (Z^*)	MAX-QBFPT (Z^*)
1	qbf020	20	151	125
2	qbf040	40	429	366
3	qbf060	60	576	[508, 576]
4	qbf080	80	1000	[843, 1000]
5	qbf100	100	[1468, 1539]	[1263,1539]
6	qbf200	200	[5385, 5826]	[3813, 5826]
7	qbf400	400	[14826, 16625]	[9645, 16625]

O *hardware* usado para testes possui uma CPU "Intel(R) Core(TM) i7-8550U (4C/8T)", com 1.80 GHz de frequência do *clock*. O computador possui 8 GB de RAM, operando com 2400 MHz. O sistema operacional utilizado foi o Zorin 15.2 (64 bits). O método foi executado com limite de 1800 segundos (30 minutos), limite de 10000 gerações, e sem limite de memória.

4. Resultados Obtidos e Análise

Os resultados obtidos nos testes são mostrados nas Tabelas 2, 3 e 4. A iteração na qual o valor máximo foi obtido é apresentada na coluna *Iter*. A corretude da geração de triplas foi testada separadamente, e os testes foram bem-sucedidos.

Os resultados foram de qualidade mista, dados os custos de referência da Tabela 1. Para as instâncias maiores, o resultado foi de ótima qualidade, ultrapassando a referência em todas as configurações, muitas vezes por uma grande margem. Nas instâncias para as quais são conhecidas soluções de custo ótimo, elas também foram alcançadas em todas as configurações. Porém, nas instâncias de tamanho médio ($n \in \{60, 80, 100\}$), houve uma grande dificuldade em alcançar os resultados conhecidos. A instância de tamanho 80 em específico foi fonte de dificuldade em conseguir bons resultados, não alcançando a referência em método algum.

Comparando o desempenho das configurações entre si, em termos de qualidade das soluções, a configuração 7 foi em todos os testes melhor ou igual que a configuração padrão, e obteve os melhores resultados para as duas maiores instâncias dentre todas as configurações. As configurações 2 a 6 também obtiveram resultados melhores ou iguais que a configuração 1 para a maior instância, mas obtiveram resultado pior que a configuração padrão para pelo menos uma outra instância. Isso indica que as estratégias alternativas *steady-state* e manutenção de diversidade tiveram um impacto positivo nos resultados, como esperado.

Em relação a tempo de execução, é notável que as soluções baseadas em elitismo chegaram ao número limite de gerações bem mais rapidamente (em alguns casos, mais de 3 vezes mais rapidamente) que as baseadas em *steady-state*. Isso indica que esse método é um dos pontos mais demorados do algoritmo, e para limites de tempo menores, essa técnica pode ser preferível.

A respeito do número de iterações, pode-se ver que em geral (com exceções), as configurações com *steady-state* encontraram suas melhores soluções em um número de gerações bem menor que as correspondentes elitistas, sendo que a maioria dessas soluções são de qualidade similar ou melhor que as elitistas. Assim, para limites menores de gerações, esse método pode ser preferível.

Comparando os resultados dos experimentos tomando em conta a variação dos parâmetros tamanho da população e taxa de mutação, de maneira geral, os experimentos que utilizaram a taxa de mutação M_2 apresentaram melhores resultados que as configurações correspondentes com taxa M_1 (configuração 5 contra a 3, e configuração 7 contra 4). Isso pode estar relacionado à proporção adequada de mutação de acordo com o tamanho da instância ($1/n$), enquanto que para M_1 a taxa é fixa para todas as instâncias.

Por fim, em relação ao tamanho da população, se compararmos as configurações 1 e 6 que utilizam o mesmo método para selecionar a população (elitismo), percebe-se que a configuração 6 atingiu o limite de gerações mais rapidamente que a configuração 1, pelo fato da população ser menor. Isto também é perceptível entre as configurações 3 e 4. Nesse último par, também é possível ver a queda (em média) na qualidade das soluções encontradas. Isso pode ser explicado pela queda de diversidade com populações menores. A combinação entre menor população e taxa

de mutação adaptativa teve bons resultados nas configurações 6 e 7, provavelmente pela maior preservação de diversidade que a maior taxa de mutação fornece.

Tabela 2: Tabela com os resultados dos testes com as configurações 1, 2 e 3.

n	Configuração 1			Configuração 2			Configuração 3		
	Valor	Iter.	Tempo (s)	Valor	Iter.	Tempo (s)	Valor	Iter.	Tempo (s)
20	125	7	7,639	125	5	26,402	125	8	40,228
40	366	84	27,177	366	19	94,592	366	42	145,267
60	495	184	60,635	503	41	209,324	495	45	319,778
80	809	99	107,032	834	66	372,714	810	79	583,796
100	1175	219	157,736	1135	1475	579,611	1192	64	902,505
200	3987	3499	594,194	3831	210	1800	3959	285	1800
400	10144	7160	1800	11482	1276	1800	11318	1122	1800

Tabela 3: Tabela com os resultados dos testes com as configurações 4 e 5.

n	Configuração 4			Configuração 5		
	Valor	Iter.	Tempo (s)	Valor	Iter.	Tempo (s)
20	125	10	20,384	125	2	46,818
40	366	29	71,964	366	32	157,871
60	430	6418	153,972	500	49	343,318
80	778	134	274,675	810	74	599,578
100	1189	392	420,406	1192	64	908,128
200	3893	1822	1661,908	3902	227	1800
400	11330	1306	1800	11738	1041	1800

Tabela 4: Tabela com os resultados dos testes com as configurações 6 e 7.

n	Configuração 6			Configuração 7		
	Valor	Iter.	Tempo (s)	Valor	Iter.	Tempo (s)
20	125	4160	6,898	125	43	21,411
40	366	46	22,071	366	27	73,894
60	508	1711	46,232	495	1030	153,331
80	826	5467	82,49	813	188	274,516
100	1189	583	119,946	1189	392	417,941
200	3983	393	460,613	4038	1492	1644,922
400	11473	2005	1800	11792	1475	1800

Referências

[1] C. R. Reeves, "Genetic algorithms," in *Handbook of Metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), vol. 146 of *International Series in Operations Research & Management Science*, ch. 5, pp. 109–139, Springer Science+Business Media, 2010.

[2] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, (1975) (1992).