Atividade 4
Genetic Algorithm para o MAX-SC-QBF

MO824 - Tópicos em Otimização Combinatória - Unicamp

Carlos E. Cavalieri Furtado
RA 219748
c219748@dac.unicamp.br

Antônio De Cesare Del Nero
RA 220277
a220277@dac.unicamp.br

Gabrielle da Silva Barbosa
RA 233862
g233862@dac.unicamp.br

I. Introdução

Este relatório apresenta a implementação de um Genetic Algorithm (GA) para o problema de maximização de função binária quadrática com restrições de cobertura de conjuntos (MAX-SC-QBF), desenvolvido para a disciplina MO824. O trabalho consiste na implementação do algoritmo base como proposto em [1]. Adicionalmente, são avaliados dois tamanhos de população (constante e em função do tamanho da instância), duas taxas de mutação (constante e em função do tamanho da instância) e estratégias evolutivas alternativas (*Alternativa 1* e *Alternativa 2*)

O relatório está estruturado em seis seções: (1) esta introdução; (2) definição formal do problema MAX-SC-QBF e sua formulação matemática como um problema de Programação Linear Inteira; (3) metodologia empregada, incluindo instâncias, implementação e ambiente computacional; (4) resultados experimentais com análise comparativa de desempenho, tempo de execução e melhoria em relação ao PLI da Atividade 1, ao GRASP da atividade 2 e ao Tabu Search da Atividade 3; (5) conclusões sobre o comportamento do modelo em diferentes configurações; e (6) referências bibliográficas utilizadas.

As tarefas foram dividas da seguinte forma entre o grupo:

- Carlos: Responsável pela implementação do Generic Algorithm padrão. Implementou também as variações de tamanhos de população (constante e em função do tamanho da instância) e taxas de mutação (constante e em função do tamanho da instância). Além disso, elaborou a estrutura do relatório e redigiu a metologia do GA padrão;
- Antônio:
- Gabrielle:

Por fim, todos os alunos participaram da discussão dos resultados e conclusão do trabalho.

II. O Problema MAX-QBF com Set Cover

A. Definição do Problema

No problema MAX-SC-QBF, desejamos maximizar uma função binária quadrática (QBF) sujeita a restrições de cobertura de conjuntos, onde o universo a ser coberto é o próprio conjunto de variáveis da QBF. Uma QBF é uma função $f : \mathbb{B}^n \to \mathbb{R}$ que pode ser expressa como uma soma de termos quadráticos:

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

onde $a_{ij} \in \mathbb{R}, (i, j = 1, ..., n)$ são os coeficientes da função f. Logo, o problema de maximização da QBF pode ser expresso como $Z = max\ f(x)$

Para o caso especial com com restrições de cobertura de conjuntos, definimos $N = \{1, \ldots, n\}$ como o *conjunto de variáveis* da QBF. Seja $S = \{S_1, \ldots, S_m\}$ uma coleção de subconjuntos $S_i \subseteq N$, representando as variáveis que o subconjunto i cobre. Cada subconjunto i está associado a uma variável binária x_i indicando sua seleção. Para cada par (i, j) de subconjuntos, temos um coeficiente $a_{ij} \in \mathbb{R}$ que representa o ganho (positivo ou negativo) por selecionar ambos simultaneamente.

Nosso objetivo é selecionar subconjuntos de forma que:

- todas as variáveis da QBF sejam cobertas, ou seja, para todo $k \in N$, exista ao menos um S_i tal que $k \in S_i$ e $x_i = 1$;
- seja maximizado o ganho quadrático total derivado das interações entre subconjuntos selecionados.

B. Formulação Matemática

Para formular o problema como um PLI, vamos primeiramente linearizar a função objetivo 1, introduzindo a variável

$$y_{ij} = \begin{cases} 1 & \text{se } x_i \text{ e } x_j \text{ são selecionados} \\ 0 & \text{caso contrário} \end{cases}$$

Dessa forma, a nova função objetivo é dada por:

$$max \quad \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot y_{ij} \tag{2}$$

Em seguida, vamos definir as restrições que garantem que para determinado conjunto $(i, j = 1, ..., n)$, $y_{i,j} = 1$ se e somente se $x_i = 1$ e $x_j = 1$:

$$y_{ij} \leq x_i \tag{3}$$
$$y_{ij} \leq x_j \tag{4}$$
$$y_{ij} \geq x_i + x_j - 1 \tag{5}$$

Por fim, adicionamos também as restrições que garantem que todas as variáveis da QBF sejam cobertas por pelo menos um subconjunto:

$$\sum_{i:k \in S_i} x_i \geq 1 \qquad \forall k \in N \tag{6}$$

III. METODOLOGIA

A. Instâncias

Para avaliar o desempenho das abordagens propostas em diferentes cenários, foram implementados três padrões distintos de geração de subconjuntos, combinados com cinco tamanhos de instância ($n \in \{25, 50, 100, 200, 400\}$), totalizando 15 instâncias de teste.

Padrão 1 - Subconjuntos Pequenos: Cada subconjunto S_i contém entre 2 e 4 elementos, iniciando sempre com o elemento i para garantir cobertura, complementado por elementos aleatórios. Este padrão gera alta fragmentação com muitas variáveis binárias, simulando problemas com opções de cobertura limitada.

Padrão 2 - Estrutura de Vizinhança: Os subconjuntos são formados por elementos consecutivos em estrutura circular, com tamanhos variando entre $n/10$ e $n/5$. Por exemplo, S_i pode conter $\{i, i+1, i+2, ...\}\ mod\ n$. Este padrão simula problemas com localidade espacial ou temporal, onde elementos próximos tendem a ser cobertos juntos.

Padrão 3 - Tamanhos Heterogêneos: Implementa uma distribuição mista com 30% de subconjuntos pequenos (2-3 elementos), 50% médios ($n/8$ a $n/4$ elementos) e 20% grandes ($n/3$ a $n/2$ elementos), com elementos selecionados aleatoriamente. Este padrão representa cenários mais realistas onde as opções de cobertura têm custos e capacidades variadas.

A matriz de coeficientes A é gerada com valores aleatórios uniformemente distribuídos no intervalo $[-10, 10]$, armazenada em formato triangular superior. Todas as instâncias utilizam "seeds" fixas para garantir reprodutibilidade dos experimentos.

Na demonstração de resultados, as instâncias terão nomenclatura $n[X]p[Y]$, onde X é o tamanho de instância e Y é o padrão de geração exposto acima.

B. O Generic Algorithm

Nesta subseção descrevemos a metaheurística baseada em GA usada no trabalho, ressaltando seu funcionamento geral e componentes principais.

*1) **Modelo padrão**:* O Genetic Algorithm implementado segue o framework clássico descrito por [1], adaptado para o problema MAX-SC-QBF. O algoritmo opera através de gerações sucessivas, aplicando operadores genéticos sobre uma população de cromossomos binários.

Codificação de uma solução: Cada solução é representada por um cromossomo binário de tamanho n, onde n é o número de subconjuntos disponíveis no problema. Cada gene i do cromossomo pode assumir valores 0 ou 1, indicando respectivamente a não seleção ou seleção do subconjunto i. A decodificação de um cromossomo para uma solução consiste em adicionar à solução todos os índices cujos genes possuem valor 1. Por exemplo, o cromossomo $[1, 0, 1, 0, 1]$ representa a seleção dos subconjuntos 0, 2 e 4.

Geração da população inicial: A população inicial de tamanho *popSize* é gerada utilizando uma estratégia mista. O primeiro cromossomo é sempre inicializado com todos os genes iguais a 1, garantindo uma solução factível inicial (embora possivelmente de baixa qualidade). Os demais cromossomos são gerados aleatoriamente, onde cada gene tem probabilidade de 0.5 de assumir o valor 1. Para cada cromossomo gerado aleatoriamente, verifica-se sua factibilidade através da decodificação e avaliação. Caso a solução seja inválida (custo infinito), todos os genes do cromossomo são ajustados para 1, assegurando factibilidade.

Quanto ao tamanho da população, foram testadas duas abordagens: população constante com tamanhos fixos de 100 ou 200 indivíduos, e população em função do tamanho da instância, calculada como $max(50, 2 \times n)$, onde n é o número de subconjuntos.

Método de seleção de indivíduos: O método implementado é a seleção por torneio binário (binary tournament selection). Para selecionar cada pai, dois indivíduos são escolhidos aleatoriamente da população corrente e aquele com maior fitness é selecionado. Este processo é repetido até que *popSize* pais sejam selecionados, formando o conjunto de pais que participarão do cruzamento.

Método de recombinação (crossover): O operador implementado é o crossover de dois pontos (2-point crossover). Para cada par de pais consecutivos na população selecionada, dois pontos de corte são escolhidos aleatoriamente no cromossomo. O segmento entre esses pontos é trocado entre os pais, gerando dois filhos. Especificamente, dados os pais $P1$ e $P2$ e os pontos de corte $c1$ e $c2$ (onde $c1 < c2$):

- Filho1 recebe genes de $P1$ nas posições $[0, c1)$, genes de $P2$ nas posições $[c1, c2)$, e genes de $P1$ nas posições $[c2, n)$
- Filho2 recebe genes de $P2$ nas posições $[0, c1)$, genes de $P1$ nas posições $[c1, c2)$, e genes de $P2$ nas posições $[c2, n)$

Método de mutação: A mutação implementada é do tipo bit-flip, aplicada gene a gene em cada cromossomo filho. Para cada posição no cromossomo, um número aleatório uniforme entre 0 e 1 é gerado. Se este número for menor que a taxa de mutação, o gene correspondente é invertido ($0 \rightarrow 1$ ou $1 \rightarrow 0$). Esta abordagem permite exploração do espaço de busca, ajudando a evitar convergência prematura.

Quanto à taxa de mutação, foram testadas duas abordagens: taxa constante com valores fixos de 0.01 ou 0.05 por gene, e taxa em função do tamanho da instância, calculada como $1/n$,

seguindo a recomendação clássica da literatura que sugere uma mutação esperada por cromossomo.

Atualização da população: O método implementado é elitista. Após a geração dos filhos através de crossover e mutação, o pior cromossomo da população de filhos é identificado. Se seu fitness for inferior ao fitness do melhor cromossomo da geração anterior, o pior filho é substituído pelo melhor cromossomo anterior, garantindo que a melhor solução encontrada até o momento seja preservada.

*2) **Estratégias evolutivas alternativas**:* Além da implementação padrão descrita acima, neste trabalho serão comparadas duas estratégias evolutivas alternativas:

- *Alternativa 1:*
- *Alternativa 2:*

*3) **Critérios de parada**:* Adotaremos como critérios de parada as seguintes condições:

- Tempo limite por experimento: 30 minutos;
- Número máximo de gerações: 1000 gerações;
- Número máximo de gerações sem melhoria: 100 gerações.

C. Implementação e Execução

Ambiente de Desenvolvimento: O modelo matemático proposto e as metaheurísticas foram implementadas em Java, utilizando o framework fornecido pelos docentes da disciplina. O IDE Eclipse foi escolhido para desenvolvimento.

Especificações Computacionais: Os experimentos foram executados com recursos locais, que apresentam as seguintes especificações técnicas:

- Modelo: MacBook Air (M1, 2020)
- Processador: Apple M1 (CPU de 8 núcleos)
- Memória RAM: 8 GB
- Sistema Operacional: macOS Sequoia 15.3.2

Disponibilidade do Código: Todo o código-fonte, incluindo a implementação da metaheurística e os scripts de execução, assim como as instâncias utilizadas nos experimentos, estão disponíveis no repositório GitHub: https://github.com/ccavalieri/MO824. Esta disponibilização permite a reprodutibilidade completa e acesso aos arquivos gerados nos experimentos.

D. Coleta de Resultados

Para avaliação dos resultados, as seguintes métricas foram selecionadas:

- Melhor solução encontrada (MS): o melhor valor encontrado através da abordagem utilizada;
- Tempo de execução: o tempo de execução até a parada do algoritmo;
- Número de iterações: o número de iterações executadas até a parada do algoritmo;
- Taxa de Melhoria: razão de melhoria em relação ao modelo exato desenvolvido na atividade 1, o GRASP na Atividade 2 ou Tabu na Atividade 3, calculado como $\frac{MS_{GA}-MS_x}{MS_x}$, onde MS_x é o melhor valor encontrado no PLI, GRASP ou TS.

IV. Resultados

V. Conclusão

Referências

[1] Reeves, C.R. (2010) “Genetic Algorithms.” In: Gendreau, M. and Potvin, J.-Y. (eds.) *Handbook of Metaheuristics*. International Series in Operations Research and Management Science, vol. 146. Springer. Available: https://doi.org/10.1007/978-1-4419-1665-5_10

1. Apêndice

3. APÊNDICE

Declaração de uso de Inteligência Artificial

Foram utilizadas ferramentas de IA para tradução de códigos entre Python e Java e para indentação, dado que alguns discentes tinham maior fluência em Python.

Também foi usada IA para produção de blocos de código auxiliares ao projeto, como por exemplo para criação da função de escrever um csv e para correções nas funções do GA (debug)

Por fim, também foi usada IA para auxílio na correção e formatação deste relatório.