Atividade 1 - MC859

Luiz Fernando Batista Morato - RA 223406
Pietro Grazzioli Golfetto - RA 223694

Agosto 2025

Este documento contém o relatório da solução da primeira atividade da disciplina MC859 - Tópicos em Otimização Combinatória para o segundo semestre de 2025.

1 Descrição do Gerador de Instâncias

As instâncias do problema MAX-SC-QBF foram geradas por meio do script `generator.py`, que produz arquivos no diretório `instances/`. O gerador utiliza como parâmetro principal o número de variáveis $n \in \{25, 50, 100, 200, 400\}$ e adota três diferentes padrões de construção dos conjuntos:

- **Padrão esparso (sparse):** cada elemento é incluído em poucos subconjuntos (entre 2 e 4), de modo a gerar instâncias menos densas.
- **Padrão denso (dense):** cada elemento pertence a muitos subconjuntos, cobrindo de 20% a 50% deles.
- **Padrão estruturado (structured):** há um pequeno número de subconjuntos grandes ("hubs"), cobrindo de 20% a 50% dos elementos, e muitos subconjuntos pequenos que cobrem apenas 2 a 5 elementos. Os hubs compõem cerca de 10% dos subconjuntos.

Além da família de subconjuntos, o gerador também cria uma matriz $A \in \mathbb{Z}^{n \times n}$, triangular superior, contendo coeficientes inteiros sorteados aleatoriamente no intervalo $[-20, 20]$. Cada instância é salva em um arquivo texto contendo:

1. O valor de n.
2. O tamanho de cada subconjunto.
3. Os elementos de cada subconjunto (indexados a partir de 1).
4. Os coeficientes da matriz A.

No total, foram geradas 15 instâncias, combinando os 5 valores de n com os 3 padrões de geração.

2 Modelo Matemático

O problema MAX-SC-QBF pode ser descrito da seguinte forma. Seja $U = \{1, 2, \ldots, n\}$ o conjunto de elementos, e $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ a família de subconjuntos gerada. A cada par de variáveis (i, j) está associado um coeficiente a_{ij} proveniente da matriz A.

Variáveis de decisão

$$x_i = \begin{cases} 1 & \text{se o subconjunto } S_i \text{ é selecionado,} \\ 0 & \text{caso contrário,} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{se } x_i = x_j = 1 \text{ (linearização do termo } x_i x_j), \\ 0 & \text{caso contrário.} \end{cases}$$

Função objetivo

Maximizar:

$$\max \sum_{i \leq j} a_{ij}\, y_{ij}.$$

Restrições

1. **Cobertura:** cada elemento $k \in U$ deve estar contido em pelo menos um subconjunto selecionado:

$$\sum_{i : k \in S_i} x_i \geq 1, \quad \forall k \in U.$$

2. **Linearização:** garantir que $y_{ij} = x_i \cdot x_j$:

$$y_{ij} \leq x_i, \quad y_{ij} \leq x_j, \quad y_{ij} \geq x_i + x_j - 1, \quad \forall (i, j).$$

3. **Domínio:**

$$x_i \in \{0, 1\}, \quad y_{ij} \in \{0, 1\}.$$

3 Especificação do Ambiente de Execução

- **Máquina:**
  - **Processador:** Intel(R) Core(TM) 7 150U
  - **Núcleos:** 10 núcleos físicos, 12 processadores lógicos
  - **Sistema Operacional:** Ubuntu 22.04.5 LTS
- **Versão do Gurobi:** 12.0.3

- **Linguagem de Programação:** Python 3.10.12
- **Estratégia do Solver:** Para lidar com a complexidade das instâncias maiores, a estratégia do solver Gurobi foi ajustada para priorizar a busca por soluções viáveis de alta qualidade. Foram configurados os parâmetros `MIPFocus=1` e `Heuristics=0.1`.

4 Análise de Resultados

As 15 instâncias foram geradas com $n \in \{25, 50, 100, 200, 400\}$ e três padrões (*sparse*, *dense*, *structured*), e resolvidas com limite de tempo de 600 s por instância. A Tabela 1 resume o valor da função objetivo, o *gap* de otimalidade e o tempo de execução.

n	padrão	valor	gap (%)	tempo (s)
25	dense	458	0.00	0.16
25	sparse	235	0.00	0.13
25	structured	535	0.00	0.05
50	dense	1306	0.00	9.74
50	sparse	1322	0.00	2.88
50	structured	1453	0.00	1.60
100	dense	3374	37.70	600.01
100	sparse	2755	36.01	600.02
100	structured	2664	51.88	600.01
200	dense	9617	191.80	600.05
200	sparse	9642	166.86	600.02
200	structured	9640	192.71	600.03
400	dense	20841	794.17	600.06
400	sparse	20155	663.32	600.07
400	structured	24214	675.79	600.08

Table 1: Resumo dos resultados por instância (gap reportado pelo Gurobi multiplicado por 100). Para $n \geq 100$, todas as execuções atingiram o limite de 600 s.

Análise do Gap de Otimalidade. O *gap* de otimalidade é a métrica que o Gurobi utiliza para medir a qualidade da solução incumbente em relação ao ótimo teórico. Ele é calculado como a diferença percentual entre o melhor limite superior (o maior valor que qualquer solução poderia teoricamente atingir) e o limite inferior (o valor da melhor solução viável encontrada).

Um gap de 0% significa que a otimalidade da solução foi provada. Conforme a Tabela 1, para $n \leq 50$, o solver conseguiu fechar o gap e provar a otimalidade em todos os casos. Para $n \geq 100$, o tempo limite foi atingido com gaps consideráveis, chegando a mais de 700% nos casos mais difíceis. Um gap elevado não implica que a solução incumbente seja ruim, mas sim que a garantia de sua qualidade é

fraca: o limite superior calculado pelo solver ainda está muito distante do valor da melhor solução encontrada.

Isso sugere que a relaxação linear do modelo, potencialmente devido ao grande número de variáveis e restrições introduzidas pela linearização, é fraca para instâncias grandes, tornando difícil para o algoritmo de branch-and-bound podar a árvore de busca eficientemente e convergir para o ótimo no tempo estipulado.

Efeito do padrão (nesta amostra). Em $n = 25$ e $n = 50$, o padrão *structured* obteve maiores valores; em $n = 100$, o *dense* foi superior; em $n = 200$ os três ficaram praticamente empatados; em $n = 400$, *structured* voltou a liderar. Contudo, como a matriz de coeficientes A da função objetivo foi gerada de forma independente para cada instância, as diferenças observadas refletem tanto a estrutura dos conjuntos S_i (que afeta as restrições) quanto a aleatoriedade dos coeficientes (que afeta a função objetivo), tornando impossível uma comparação causal direta.

Uma análise mais robusta exigiria um desenho experimental diferente: para um dado valor de n, uma única matriz A seria gerada. Em seguida, os três padrões seriam aplicados para criar três famílias distintas de subconjuntos, $\mathcal{S}_{\text{esparso}}$, $\mathcal{S}_{\text{denso}}$ e $\mathcal{S}_{\text{estruturado}}$. Isso resultaria em três instâncias que compartilham a mesma função objetivo, mas diferem em suas restrições de cobertura. Tal abordagem isolaria o impacto da estrutura dos conjuntos na dificuldade de se encontrar soluções viáveis e ótimas para uma mesma paisagem de otimização.

Disponibilidade do Código

O código-fonte, o gerador de instâncias e as 15 instâncias utilizadas neste trabalho estão disponíveis publicamente no seguinte repositório do GitHub:

`https://github.com/fernandomorato/mc859/tree/master/atividade-01`

5 Conclusão

Nesta atividade, o problema MAX-SC-QBF foi modelado com sucesso utilizando programação linear inteira e resolvido através de uma implementação em Python com o solver Gurobi. Para avaliar o desempenho do modelo, foi desenvolvido um gerador de instâncias capaz de criar problemas com diferentes tamanhos e estruturas de cobertura.

Os resultados experimentais demonstraram a viabilidade do modelo para instâncias de pequeno porte ($n \leq 50$), que foram resolvidas otimamente em poucos segundos. Para instâncias maiores ($n \geq 100$), a complexidade computacional do problema se tornou evidente, com todas as execuções atingindo o limite de tempo de 10 minutos e resultando em gaps de otimalidade significativos, o que corrobora a natureza NP-difícil do problema. A estratégia de

ajuste de parâmetros do Gurobi (`MIPFocus`, `Heuristics`) foi fundamental para a obtenção de soluções incumbentes de qualidade nos casos mais difíceis.

Por fim, a análise revelou que o método de geração de instâncias adotado impede uma comparação causal direta entre os padrões de cobertura. Como trabalho futuro, sugere-se um desenho experimental aprimorado para isolar este efeito: para cada n, gerar uma única matriz A e, a partir dela, criar três instâncias distintas, uma para cada padrão de conjuntos. Esta metodologia permitiria uma análise conclusiva sobre como a densidade e a estrutura das restrições de cobertura impactam o desempenho do solver para uma mesma função objetivo.