Celso Gabriel Ferreira Marçal Prado - 235393

Gabrielle da Silva Barbosa - 233862

Marcelo de Souza Corumba de Campos - 236730

22 de agosto de 2025

MO824/MC859 Atividade 1

Introdução

A atividade proposta consistiu na modelagem e solução de instâncias do problema MAX-SC-QBF, que consiste em uma adaptação do MAX-QBF, NP-difícil, adicionando uma nova restrição de cobertura de conjuntos. O problema visa maximizar a solução da função f(x), onde a_{ij} são elementos de uma matriz A, de dimensão n x n, e x é um vetor de variáveis binárias.

$$f(x) = \sum_{i=0}^{n} \sum_{j=0}^{n} a_{i,j} x_i x_j$$

Além disso, essas variáveis x_i devem ser submetidas a uma cobertura de conjuntos, de forma que há o conjunto N = {1,…,n} de variáveis da QBF, e subconjuntos S = {S_1,...,S_n}, tal que $S_i \subseteq N$. Um subconjunto S_i é escolhido caso $x_i = 1$, e a posição a_{ij} da matriz A representa, portanto, o ganho (positivo ou negativo) ao escolher os conjuntos S_i e S_j. A união de todos os subconjuntos selecionados deve conter todos os elementos do conjunto N.

Modelo Matemático

O problema então foi modelado na forma de PLI (Programação Linear Inteira). Para linearizar o problema, visto que há uma multiplicação de variáveis que não seria permitida, foi

criada uma nova variável y_{ij}, que representa uma multiplicação $x_i x_j$. Como x_i são variáveis binárias, é possível garantir o valor de y_{ij} utilizando um conjunto de três restrições:

$y_{i,j} \leq x_i \; \forall i,j$ (se x_i for 0, y_{ij} é 0 e, se x_i for 1, y_{ij} pode ser 1);

$y_{i,j} \leq x_j \; \forall i,j$ (análogo ao anterior, para o índice j);

$y_{i,j} \geq x_i + x_j - 1 \; \forall i,j$ (garante que, se x_i e x_j forem 1, então y_{ij} deve ser 1).

Segue abaixo o modelo completo:

1. Variáveis:

   $x_i \in \{0,1\}$, i ∈ {1,...,n}

   $y_{i,j} \in \{0,1\}$, i,j ∈ {1,...,n}

2. Função Objetivo:

   $$max \sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j} y_{i,j}$$

3. Restrições:

   $y_{i,j} \leq x_i$

   $y_{i,j} \leq x_j$

   $y_{i,j} \leq x_i + x_j - 1$

   $\sum_{i:k \in S_i} x_i \geq 1, \;\; \forall \, k \epsilon N$ (Set Cover)

Gerador de Instâncias

Os subconjuntos S foram gerados em três padrões distintos, todos os métodos garantindo que os elementos do conjunto N estejam distribuídos dentre os subconjuntos S integralmente e, consequentemente, garantindo que haja solução viável para o problema.

1. Estruturado:

   Cada subconjunto S é gerado com exatamente k elementos, de forma estruturada fazendo com que cada subconjunto possua uma sequência de k elementos consecutivos.

2. Aleatório

   Gera subconjuntos de tamanho aleatório com elementos aleatórios, com tamanho limitado em 25% do conjunto N (para garantir que não haja muitos conjuntos grandes, o que o aproximaria do problema MAX-QBF clássico). Para garantir que a união dos subconjuntos dê o conjunto N, os elementos faltantes (se houver) são adicionados a subconjuntos aleatórios, assim mantendo a completa aleatoriedade.

3. Ocorrência Controlada

   Gera um vetor de ocorrências para controlar a ocorrência de cada variável, então distribui aleatoriamente os elementos nos subconjuntos garantindo que um mesmo subconjunto não receba a mesma variável mais de uma vez. O tamanho dos subconjuntos, nesse caso, são limitados em 50% do tamanho de N.

A matriz A, por sua vez, foi gerada aleatoriamente, respeitando um intervalo definido de valores, triangular superior, como estipulado pelo problema, e garantindo a existência de pelo menos um valor negativo, de forma que não haja solução trivial no problema (ativar todos os x_i).

A escolha de manter a matriz A igual para as diferentes variações de subconjuntos foi com o intuito de que fosse possível comparar melhor a influência de cada padrão de subconjunto no desempenho e resultado final. Também foram criadas outras duas formas de geração da matriz A, uma esparsa e a outra completamente negativa, que estão disponíveis no código, mas decidimos não utilizar como comparativo pelo motivo citado.

Ambiente de Execução

Os experimentos foram executados em uma máquina com Windows 11, equipada com processador AMD Ryzen 5 5600G, que conta com 6 núcleos e 12 threads. O modelo foi implementado na linguagem Python, versão 3.13.5, utilizando a biblioteca Gurobi, versão 12.0.3. Todo o código está disponível na pasta "ativ01" do repositório, no GitHub de link https://github.com/GabrielleBarbosa/mc859.

Foram desenvolvidos três programas principais. O primeiro, "generator.py", é responsável pela criação das instâncias, recebendo como parâmetro o tipo de geração, o valor k e o tamanho n para gerar arquivos na pasta "data" que servirão como instâncias do problema. Como essas instâncias são armazenadas em arquivos, isso garante a reprodutibilidade, ou seja, é possível reproduzir exatamente os mesmos experimentos, uma vez que, dado a mesma matriz A e os mesmo subconjuntos $S = \{S_1,...,S_n\}$, os resultados obtidos serão idênticos aos relatados. O segundo programa, "main.py", consiste na implementação do modelo matemático em si, que recebe como entrada uma instância e executa a tentativa de solução com o Gurobi. Por fim, o programa "runner.py" é um auxiliar para execução de todos os 15 cenários gerados automaticamente, com limite de tempo de 10 minutos por solução. Ele percorre o diretório

“data” e executa sequencialmente o programa “main.py” para cada instância, interrompendo a sua execução pelo limite de tempo, se necessário, e armazenando os registros de execução na pasta “logs”.

Resultados do Experimento

As instâncias foram geradas visando um modo de possibilitar uma melhor comparação entre os diferentes padrões de geração de S. Para cada $n \in \{20, 50, 100, 200, 400\}$, foi gerada uma mesma matriz A, com valores aleatórios no intervalo [-100, 100], e variados os três modelos de S. Para a instância estruturada, o parâmetro k recebeu valor 4, ou seja, subconjuntos pequenos, com a expectativa de dificultar a solução, pois essa restrição fica forte visto que uma união de muitos subconjuntos é necessária para satisfazê-la.

Os resultados dos experimentos estão organizados nas Tabelas 1 a 5. Cada tabela apresenta o valor da solução obtida, o gap de otimalidade e o tempo de execução. O valor da solução corresponde ao melhor valor da função objetivo encontrado pelo solver. O gap de otimalidade representa a diferença percentual entre o melhor limite inferior e o melhor limite superior obtidos até o final da execução; quando igual a 0%, significa que a otimalidade foi provada. O tempo corresponde ao total de execução até encontrar a solução ótima ou até atingir o limite de 600 segundos.

Tabela 1 – Resultados para n=25

	Valor da Solução	Gap de Otimalidade	Tempo
Estratégia Estruturada	1668	0,00 %	0,20 s
Estratégia Aleatória	1316	0,00 %	0,14 s
Estratégia Controlada	1272	0,00 %	0,13 s

Tabela 2 – Resultados para n=50

	Valor da Solução	Gap de Otimalidade	Tempo
Estratégia Estruturada	5618	0,00 %	7,78 s
Estratégia Aleatória	5257	0,00 %	8,06 s
Estratégia Controlada	5557	0,00 %	3,87 s

Tabela 3 – Resultados para n=100

	Valor da Solução	Gap de Otimalidade	Tempo
Estratégia Estruturada	16507	36,15 %	600 s (timeout)
Estratégia Aleatória	16467	41,23 %	600 s (timeout)
Estratégia Controlada	16461	38,19 %	600 s (timeout)

Tabela 4 – Resultados para n=200

	Valor da Solução	Gap de Otimalidade	Tempo
Estratégia Estruturada	47270	209,01 %	600 s (timeout)
Estratégia Aleatória	48024	205,14 %	600 s (timeout)
Estratégia Controlada	47636	201,25 %	600 s (timeout)

Tabela 5 – Resultados para n=400

	Valor da Solução	Gap de Otimalidade	Tempo

Estratégia Estruturada	91398	927,85 %	600 s (timeout)
Estratégia Aleatória	92525	924,58 %	600 s (timeout)
Estratégia Controlada	92150	894,68 %	600 s (timeout)

A análise dos resultados mostra que, para instâncias de menor porte (n=25 e n=50), o *solver* foi capaz de encontrar a solução ótima em menos de dez segundos, com gap de otimalidade igual a 0%. Nessas situações, a estratégia estruturada apresentou os maiores valores de solução, seguida pela controlada, enquanto a aleatória apresentou desempenho inferior.

A partir de n=100, não foi possível alcançar a solução ótima no limite de tempo de 10 minutos. Ainda assim, o solver conseguiu gerar soluções viáveis, com gaps de otimalidade variando entre 36% e 42%. Esse aumento do gap está relacionado diretamente ao crescimento do espaço de busca: quanto maior n, maior o número de combinações de subconjuntos que precisam ser exploradas.

Para n=200 e n=400, os gaps se tornaram bastante elevados, chegando a ultrapassar 900% na maior instância. Isso indica que, embora o solver tenha encontrado soluções viáveis, não foi possível aproximar-se significativamente do ótimo dentro do tempo limite.

Em termos comparativos, a estratégia estruturada tende a gerar instâncias com soluções de maior valor, mas também mais desafiadoras para o solver, enquanto a aleatória apresenta soluções de menor valor e gaps consistentemente mais altos. A estratégia controlada situa-se em

posição intermediária, apresentando valores de solução próximos à estruturada, mas com gaps ligeiramente menores em algumas instâncias de maior porte.

Conclusão

Foi possível concluir o objetivo da atividade, modelando e analisando experimentalmente o comportamento da solução em diferentes situações. O grupo exercitou criatividade pensando em maneiras de gerar as instâncias do problema e testar e combinar parâmetros, de forma a buscar uma certa variabilidade nos resultados. Com tantas possibilidades, apenas três métodos de geração de instâncias acabaram sendo poucos, e seria interessante um estudo mais aprofundado, tanto com os outros métodos para a matriz A mencionados, quanto explorando outros parâmetros e/ou técnicas para os subconjuntos.

Além disso, os experimentos evidenciaram a complexidade do problema MAX-SC-QBF, mostrando que, mesmo para valores moderados de n, a resolução ótima pode demandar tempos significativos. A análise comparativa das estratégias permitiu observar padrões de desempenho, sugerindo que o tipo de geração dos subconjuntos influencia diretamente a dificuldade da instância.

Por fim, este trabalho evidencia a importância de estudar diferentes abordagens de modelagem e geração de instâncias, servindo como base para futuras pesquisas que busquem melhorar a eficiência do solver ou explorar heurísticas e métodos aproximados para problemas de grande dimensão.

Referências

Gurobi Optimization, LLC. (2025). *Gurobi Optimizer Reference Manual*.

Disponível em: https://docs.gurobi.com/projects/examples/en/current/index.html