Universidade Estadual de Campinas (Unicamp)

MO824 / MC859 – Tópicos em Otimização Combinatória

Relatório da Atividade 2 – GRASP para MAX-SC-QBF

Autores:

Everton Romanzini Colombo	RA: 257234
João Augusto Pimentel Barbosa	RA: 248341
Pedro Henrique Pinheiro Gadêlha	RA: 186985

Campinas, SP
2025

Resumo

Este relatório apresenta a implementação e avaliação de uma metaheurística GRASP (Greedy Randomized Adaptive Search Procedure) para o problema MAX-SC-QBF em cinco configurações, variando método de busca, parâmetro α da heurística de construção tradicional, assim como o método de construção em si. As diferenças concentraram-se no tempo, com o *first-improving* com $\alpha_2 = 0.8$ e construção *sampled greedy* sendo a escolha mais eficiente sem perda de qualidade neste conjunto de testes. Código, instâncias e resultados encontram-se no seguinte repositório: https://github.com/Everton-Colombo/MO824-A2-py.

1 Introdução

O GRASP é uma metaheurística reconhecida para problemas de otimização combinatória: a cada iteração, uma solução é construída de modo guloso-aleatório a partir de uma Lista Restrita de Candidatos (RCL), e então submetida a uma busca local [2]. Neste experimento, aplicamos GRASP ao MAX-SC-QBF, problema no qual se busca maximizar uma QBF sob restrições de cobertura de conjuntos (cada elemento do universo deve ser coberto por pelo menos um subconjunto selecionado).

Em nossos casos, avaliamos duas políticas de busca local (*first-improving* e *best-improving*), dois valores do parâmetro α da heurística de construção tradicional ($\alpha_1 = 0.3$ e $\alpha_2 = 0.8$) e três heurísticas construtivas (tradicional, gulosa-aleatória e sampled greedy), observando valor objetivo, cobertura da solução, tempo tomado e razão de parada.

2 Descrição do problema e modelo

Seja $N = \{1, \ldots, n\}$ o conjunto de variáveis binárias e $S = \{S_1, \ldots, S_n\}$ uma coleção de subconjuntos $S_i \subseteq N$ que definem a cobertura. A função objetivo é quadrática:

$$f(x) \;=\; \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}\, x_i x_j \,,$$

e o MAX-SC-QBF consiste em

$$\max\; f(x) \quad \text{sujeito a} \quad \sum_{i:\, k \in S_i} x_i \geq 1 \;\; \forall k \in N, \;\; x_i \in \{0,1\}.$$

Trata-se de um problema NP-difícil mesmo sem restrições adicionais [1]. As instâncias empregadas seguem o formato de matriz A triangular superior com inteiros, e família S de subconjuntos de cobertura.

3 Metodologia

3.1 Funções de avaliação e cobertura

Seja $X \subseteq \{1, \ldots, n\}$ o conjunto de subconjuntos selecionados e $A \in \mathbb{R}^{n \times n}$ a matriz de coeficientes (parte superior triangular fornecida). A função objetivo usada no GRASP é avaliada diretamente por

$$f(X) \;=\; \sum_{i \in X} \sum_{j \in X} A_{ij}.$$

Para suportar busca local eficiente, calculamos deltas incrementais:

$$\Delta^{\text{ins}}(e \mid X) \;=\; A_{ee} \;+\; \sum_{j \in X,\, j \neq e} \big(A_{ej} + A_{je}\big), \qquad \Delta^{\text{rem}}(e \mid X) \;=\; -\,\Delta^{\text{ins}}(e \mid X),$$

e, para trocas,

$$\Delta^{\text{chg}}(e_{\text{in}}, e_{\text{out}} \mid X) \;=\; \Delta^{\text{ins}}(e_{\text{in}} \mid X) - \Delta^{\text{ins}}(e_{\text{out}} \mid X) - \big(A_{e_{\text{in}}\, e_{\text{out}}} + A_{e_{\text{out}}\, e_{\text{in}}}\big).$$

A viabilidade é definida pela cobertura total do universo $\{1, \ldots, n\}$ pelos subconjuntos escolhidos: denotando por S_i o conjunto de elementos cobertos por i, a cobertura é

$$\text{cov}(X) \;=\; \frac{\big|\bigcup_{i \in X} S_i\big|}{n}, \quad \text{e a solução é viável sse } \text{cov}(X) = 1.$$

Durante a construção e o reparo, mede-se a contribuição de cobertura de um candidato $e \notin X$ por

$$\delta^{\text{cov}}(e \mid X) \;=\; \frac{|\, S_e \setminus \bigcup_{i \in X} S_i \,|}{n}.$$

3.2 Construção guloso-aleatória e variações

Empregamos três métodos de construção:

1. **Traditional:** a cada passo, calcula-se Δ^{ins} de todos os candidatos na *candidate list* (CL) e define-se um limiar por

   $$\theta(\alpha) \;=\; \min \Delta \;+\; \alpha\,(\max \Delta - \min \Delta), \qquad \alpha \in [0, 1].$$

   A *restricted candidate list* (RCL) contém os e tais que $\Delta^{\text{ins}}(e \mid X) \geq \theta(\alpha)$ *e* $\delta^{\text{cov}}(e \mid X) > 0$. Escolhe-se uniformemente um elemento da RCL e adiciona-se a X. O processo repete até $\text{cov}(X) = 1$ ou esvaziar a RCL.
2. **Random+Greedy:** inicia com uma *semente* aleatória de $\lfloor p\, n \rfloor$ elementos (sem reposição), com $p \in (0, 1]$. Em seguida, prossegue de forma *puramente gulosa*: em cada passo escolhe-

se o candidato com maior Δ^{ins} *dentre os que também aumentam a cobertura* ($\delta^{\text{cov}} > 0$). O passo só é aplicado se o melhor ganho for positivo; caso contrário, a construção é interrompida.

3. **Sampled Greedy:** em cada passo, amostra-se sem reposição um subconjunto $R \subseteq \text{CL}$ com $|R| = \min\{|\text{CL}|, \lfloor p\,n \rfloor\}$ (fração $p \in (0, 1]$) e escolhe-se o melhor elemento de R segundo Δ^{ins}. Aplica-se a adição apenas se o melhor ganho for positivo; caso contrário, a construção para. (Se a construção encerrar sem viabilidade, o reparo descrito abaixo é acionado.)

Observações práticas. (i) No *traditional*, a RCL pode aceitar deltas não positivos (se todos forem negativos), permitindo avançar rumo à cobertura total; a exigência $\delta^{\text{cov}} > 0$ evita inclusões que não progridem na viabilidade. (ii) Em *random+greedy* e *sampled greedy*, a construção cessa se não houver melhoria de objetivo; por isso, um reparo de viabilidade pode ser necessário ao final.

3.3 Reparo de viabilidade

Se a solução construída não for viável, aplica-se um reparo guloso de cobertura: enquanto $\text{cov}(X) < 1$, adiciona-se o candidato $e \in \text{CL}$ que maximiza $\delta^{\text{cov}}(e \mid X)$. Esse procedimento termina quando a cobertura atinge 100% (ou reporta falha se nenhum candidato puder aumentar a cobertura).

3.4 Busca local e vizinhanças

A busca local opera sobre três movimentos com checagem explícita de viabilidade:

- **Inserção**: adicionar $e \notin X$ com ganho $\Delta^{\text{ins}}(e \mid X)$;
- **Remoção**: remover $e \in X$ com ganho $\Delta^{\text{rem}}(e \mid X)$, apenas se a solução resultante permanecer viável;
- **Troca**: substituir $e_{\text{out}} \in X$ por $e_{\text{in}} \notin X$ com ganho Δ^{chg}, respeitando viabilidade.

Há dois modos:

- **First-improving**: percorre inserções → remoções → trocas e aplica o primeiro movimento com ganho > 0, reiniciando o ciclo até estagnação;
- **Best-improving**: avalia todos os movimentos viáveis e aplica o de maior ganho positivo, repetindo até não haver melhoria.

3.5 Critérios de parada do GRASP

O laço externo (`solve`) itera até ocorrer um dos eventos:

- **Tempo-limite**: `time_limit_secs` (nos experimentos, 1800 s (30 min) por instância);
- **Paciência**: interrompe quando não há melhoria da melhor solução global por `patience` iterações consecutivas (usamos 1000). A cada iteração, se a solução corrente melhora a melhor global, a paciência é reiniciada; caso contrário, é decrementada.;
- **Limite de iterações**: `max_iterations`, quando configurado.

3.6 Pseudocódigo

Algorithm 1 GRASP para MAX-SC-QBF

$X^\star \leftarrow \varnothing$, $f^\star \leftarrow -\infty$
while não atingiu `time_limit` **e** `patience` > 0 **e** iterações $<$ `max_iterations` **do**
  $X \leftarrow$ Construtivo(`traditional` | `random_plus_greedy` | `sampled_greedy`; parâmetros α ou p)
  if $\mathrm{cov}(X) < 1$ **then**
    $X \leftarrow$ FixSolution(X)
  end if
  $X \leftarrow$ BuscaLocal(X, modo $\in \{$`first_improve`, `best_improve`$\}$)
  if $f(X) > f^\star$ **then**
    $X^\star \leftarrow X$, $f^\star \leftarrow f(X)$, `patience` $\leftarrow$ valor_inicial
  else
    `patience` $\leftarrow$ `patience` -1
  end if
end while
return $X^\star$

3.7 Ambiente de execução

SO	Ubuntu 22.04.5 LTS (64 bits)	**Processador**	Intel Core i5-13450HX
Threads (fís./lóg.)	16 / 16 (usando até 16)	**Linguagem**	Python 3.12
Tempo-limite por instância	1800 s (30 min)	**Critério adicional**	Paciência = 1000 iterações sem melhoria

4 Resultados

As tabelas completas estão agrupadas ao final do relatório (Seção 7). Em todas as execuções, a **cobertura foi 100%** e os **valores de objetivo** observados por instância foram praticamente iguais entre as cinco configurações; as diferenças concentraram-se no **tempo** até estabilizar a

melhor solução.

Resumo comparativo das configurações

Tabela 1: Resumo comparativo das configurações (tempos em segundos).

Configuração	Tempo médio	Mediana	Timeout (%)
PADRÃO	387.91	15.02	13.3
PADRÃO+ALPHA	418.07	**49.70**	13.3
PADRÃO+BEST	**1453.37**	857.76	**46.7**
PADRÃO+HC1	394.86	19.79	13.3
PADRÃO+HC2	**384.65**	50.11	13.3

Observação: alguns tempos medidos foram maiores que 1800 s devido a sobrecusto de loop/log e finalização.

5 Discussão

Em todas as execuções, a **cobertura** foi de 100% e os **melhores valores de objetivo** por instância permaneceram, na prática, idênticos entre as cinco configurações, inclusive quando houve *timeout*. Assim, o principal fator de diferenciação observado foi o **tempo** necessário para estabilizar a melhor solução (cf. Tabelas da Seção 7).

Comparação com o Gurobi. De imediato, notou-se que o GRASP foi capaz de atingir valores melhores da função objetivo em tempos consideravelmente menores do que os obtidos na atividade 1, na qual utilizou-se o solver Gurobi. Para referência, os resultados da atividade 1 para essas mesmas instâncias encontram-se em: https://github.com/Everton-Colombo/MO824-A1/tree/main/results.

Modo de busca local. O *best-improving* elevou substancialmente o custo por iteração ao avaliar sistematicamente todos os movimentos viáveis, resultando nas maiores durações e na maior taxa de *timeouts*, sobretudo para $n \in \{200, 400\}$. O *first-improving*, por sua vez, manteve tempos consistentemente menores sem perda de qualidade final, sendo, portanto, preferível neste conjunto de instâncias.

Efeito do parâmetro α na construção *traditional*. Comparando PADRÃO+$\alpha_1 = 0.3$ e PADRÃO+$\alpha_2 = 0.8$ (ambos com *first-improving*), verificou-se impacto de tempo sensível, porém dependente da instância: $\alpha_1 = 0.3$ apresentou *mediana* de tempo menor (execuções típicas mais rápidas), enquanto $\alpha_2 = 0.8$ foi, ocasionalmente, mais veloz em instâncias grandes específicas (por exemplo, geração 3 com $n = 400$). Em suma, o melhor α é dependente instância.

Heurísticas construtivas alternativas. As variantes *Random+Greedy* e *Sampled Greedy* preservaram a mesma qualidade final do PADRÃO e reduziram o custo em vários cenários. A *Sampled Greedy* destacou-se pela *média* de tempo global mais baixa, graças à avaliação de um subconjunto amostrado de candidatos a cada passo; já a *Random+Greedy* mostrou-se competitiva e, em algumas instâncias de $n = 200$, foi a mais rápida. O eventual reparo de viabilidade ao fim da construção não alterou essa tendência.

Escalabilidade e efeito da geração. O custo cresce notavelmente com n. Para $n \leq 100$, todas as configurações terminam com folga de tempo; em $n = 200$, os tempos aumentam de forma marcante; e, em $n = 400$, surge clara dependência da *estrutura* da geração: na geração 2, todas as variantes atingiram *timeout*; já na geração 3, as variantes com *first-improving* concluíram antes do limite, enquanto o *best-improving* voltou a exceder o tempo. Logo, a estrutura das instâncias é tão determinante quanto o tamanho.

No *trade-off* tempo/qualidade, o *first-improving* aliado a uma construção eficiente (*Sampled Greedy* ou PADRÃO com α moderado) foi a melhor escolha. O *best-improving* não apresentou ganhos de qualidade que compensassem o maior custo, e a escolha de α deve ser ajustada por instância.

6 Conclusão

Os experimentos mostraram que todas as configurações alcançam 100% de cobertura e valores de objetivo praticamente equivalentes por instância; assim, o fator decisivo é o tempo. O modo *first-improving* revelou-se mais eficiente e com menor incidência de *timeouts*, enquanto o *best-improving* aumentou o custo por iteração sem ganhos de qualidade. Entre os construtivos, *Sampled Greedy* e *Random+Greedy* preservaram a qualidade e, com frequência, reduziram o tempo face ao método tradicional; em instâncias maiores, a dificuldade estrutural elevou o risco de *timeout*, mas ainda com vantagem em relação ao *first-improving*. O parâmetro α influenciou o custo da qualidade da solução inicial, porém sem um valor universalmente dominante; escolhas moderadas mostraram-se mais robustas.

Em termos práticos, recomendamos priorizar *first-improving* combinado a uma construção eficiente (*Sampled Greedy* ou *Random+Greedy*) e utilizar um parâmetro α moderado quando a construção tradicional for empregada. Para instâncias muito grandes, vale investigar *Reactive GRASP* (autoajuste de α), *path-relinking* entre soluções de elite e otimizações de avaliação incremental (armazenamento de somatórios/deltas) com vizinhanças ampliadas ou estratificadas, visando reduzir o custo por iteração e melhorar a escalabilidade de tempo.

7 Tabelas

Configuração 1 – PADRÃO ($\alpha_1 = 0.3$, first-improving, traditional)

Tabela 2: Resultados da Configuração 1 (PADRÃO, $\alpha_1 = 0.3$)

instance	n	stop_reason	best_objective	coverage	time_taken (s)
1.1	200	patience_exceeded	4614.25	1.0	616.3952
1.2	200	patience_exceeded	4668.07	1.0	346.2864
1.3	50	patience_exceeded	645.62	1.0	9.7015
1.4	25	patience_exceeded	261.06	1.0	0.9726
1.5	100	patience_exceeded	1525.48	1.0	84.2195
2.1	50	patience_exceeded	575.87	1.0	6.3606
2.2	400	time_limit	14472.64	1.0	1800.4601
2.3	25	patience_exceeded	210.16	1.0	1.1799
2.4	400	time_limit	13589.31	1.0	1800.6460
2.5	100	patience_exceeded	1550.49	1.0	60.8886
3.1	400	patience_exceeded	238058.87	1.0	898.9136
3.2	100	patience_exceeded	14887.43	1.0	15.0177
3.3	200	patience_exceeded	59765.97	1.0	176.1821
3.4	25	patience_exceeded	981.05	1.0	0.6840
3.5	25	patience_exceeded	766.71	1.0	0.6799

Configuração 2 – PADRÃO+ALPHA ($\alpha_2 = 0.8$, first-improving, traditional)

Tabela 3: Resultados da Configuração 2 (PADRÃO, $\alpha_2 = 0.8$)

instance	n	stop_reason	best_objective	coverage	time_taken (s)
1.1	200	patience_exceeded	4614.25	1.0	1118.6227
1.2	200	patience_exceeded	4668.07	1.0	536.1323
1.3	50	patience_exceeded	645.62	1.0	5.8153
1.4	25	patience_exceeded	261.06	1.0	1.0809
1.5	100	patience_exceeded	1525.35	1.0	111.2763
2.1	50	patience_exceeded	575.87	1.0	6.1729
2.2	400	time_limit	14472.64	1.0	1803.5759
2.3	25	patience_exceeded	210.16	1.0	0.9017
2.4	400	time_limit	13589.31	1.0	1800.8252
2.5	100	patience_exceeded	1550.49	1.0	49.6975
3.1	400	patience_exceeded	238058.87	1.0	720.4827
3.2	100	patience_exceeded	14887.43	1.0	14.0803
3.3	200	patience_exceeded	59765.97	1.0	107.3665
3.4	25	patience_exceeded	981.05	1.0	0.9212
3.5	25	patience_exceeded	766.71	1.0	0.5326

Configuração 3 – PADRÃO+BEST ($\alpha_1 = 0.3$, *best-improving*, tradicional)

Tabela 4: Resultados da Configuração 3 (PADRÃO+BEST)

instance	n	stop_reason	best_objective	coverage	time_taken (s)
1.1	200	time_limit	4614.25	1.0	1801.0888
1.2	200	time_limit	4668.07	1.0	1805.2023
1.3	50	patience_exceeded	645.62	1.0	82.6778
1.4	25	patience_exceeded	261.06	1.0	5.1376
1.5	100	patience_exceeded	1525.48	1.0	857.7602
2.1	50	patience_exceeded	575.87	1.0	47.5807
2.2	400	time_limit	14339.26	1.0	1871.4853
2.3	25	patience_exceeded	210.16	1.0	5.2978
2.4	400	time_limit	13536.58	1.0	1865.6972
2.5	100	patience_exceeded	1550.49	1.0	760.4555
3.1	400	time_limit	238058.87	1.0	2076.7661
3.2	100	time_limit	14887.43	1.0	1800.3975
3.3	200	time_limit	59765.97	1.0	1805.2419
3.4	25	patience_exceeded	981.05	1.0	6.7510
3.5	25	patience_exceeded	766.71	1.0	7.8043

Configuração 4 – PADRÃO+HC1 ($\alpha_1 = 0.3$, first-improving, random plus greedy)

Tabela 5: Resultados da Configuração 4 (PADRÃO+HC1 – Random+Greedy)

instance	n	stop_reason	best_objective	coverage	time_taken (s)
1.1	200	patience_exceeded	4614.25	1.0	486.8412
1.2	200	patience_exceeded	4668.07	1.0	682.9842
1.3	50	patience_exceeded	645.62	1.0	5.2578
1.4	25	patience_exceeded	261.06	1.0	1.5457
1.5	100	patience_exceeded	1525.48	1.0	70.3684
2.1	50	patience_exceeded	575.87	1.0	11.6622
2.2	400	time_limit	14472.64	1.0	1800.1754
2.3	25	patience_exceeded	210.16	1.0	0.9513
2.4	400	time_limit	13589.31	1.0	1802.4110
2.5	100	patience_exceeded	1550.49	1.0	72.5960
3.1	400	patience_exceeded	238058.87	1.0	851.4960
3.2	100	patience_exceeded	14887.43	1.0	19.7919
3.3	200	patience_exceeded	59765.97	1.0	115.6875
3.4	25	patience_exceeded	981.05	1.0	0.6568
3.5	25	patience_exceeded	766.71	1.0	0.4302

Configuração 5 – PADRÃO+HC2 ($\alpha_1 = 0.3$, first-improving, sampled greedy)

Tabela 6: Resultados da Configuração 5 (PADRÃO+HC2 – Sampled Greedy)

instance	n	stop_reason	best_objective	coverage	time_taken (s)
1.1	200	patience_exceeded	4614.25	1.0	581.4256
1.2	200	patience_exceeded	4668.07	1.0	318.4235
1.3	50	patience_exceeded	645.62	1.0	7.3334
1.4	25	patience_exceeded	261.06	1.0	1.1575
1.5	100	patience_exceeded	1525.35	1.0	50.1105
2.1	50	patience_exceeded	575.87	1.0	9.2088
2.2	400	time_limit	14469.84	1.0	1801.0542
2.3	25	patience_exceeded	210.16	1.0	0.9696
2.4	400	time_limit	13589.31	1.0	1800.1091
2.5	100	patience_exceeded	1550.49	1.0	88.7145
3.1	400	patience_exceeded	238058.87	1.0	922.0314
3.2	100	patience_exceeded	14887.43	1.0	23.8773
3.3	200	patience_exceeded	59765.97	1.0	164.2714
3.4	25	patience_exceeded	981.05	1.0	0.6802
3.5	25	patience_exceeded	766.71	1.0	0.3644

8 Distribuição de tarefas

Everton Romanzini Colombo (257234): Implementação em código. Tradução inicial para Python, heurística de construção sampled greedy e método de busca best improving. Revisão geral do código. Execução dos experimentos.

João Augusto Pimentel Barbosa (248341): Implementação em código. Versões iniciais da heurística de construção random plus greedy e método de busca best improving.

Pedro Henrique Pinheiro Gadêlha (186985): Redigiu o relatório e participou do planejamento do projeto.

Referências

[1] Kochenberger, G., et al. *The unconstrained binary quadratic programming problem: a survey.* Journal of Combinatorial Optimization, 28:58–81, 2014.

[2] Resende, M.G.C.; Ribeiro, C.C. *Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications.* In: Gendreau, M.; Potvin, J.-Y. (eds.) *Handbook of Metaheuristics*, Springer.