Submetido para MO824

Atividade 8 - Grupo 4 - MO824

Victor Ferreira Ferrari - RA 187890
Luma Gabino Vasconcelos - RA 202495
Guilherme de Oliveira Macedo - RA 208395

Resumo. Este trabalho apresenta avaliações de desempenho de três metaheurísticas para o problema MAX-QBFPT, além da resolução de dois modelos matemáticos distintos pelo *software* Gurobi, por *TTT Plots* (apenas as metaheurísticas) e Perfis de Desempenho, junto com uma justificativa matemática por trás de cada método. As metaheurísticas testadas foram GRASP, Busca Tabu e Algoritmo Genético. Os resultados indicam que os métodos, em geral, não seguem distribuições exponenciais deslocadas, com exceção do Algoritmo Genético, e um destaque para a Busca Tabu por conseguir ótimos resultados de modo eficiente, em menos tempo que os outros métodos.

Palavras-chave. Otimização, Desempenho, ttt-plots, performance profiles, QBFPT, QBF, metaheurísticas.

1. Introdução

Este trabalho consiste na análise de desempenho de soluções exatas e heurísticas, baseada nos métodos de perfis de desempenho e *ttt-plots*, para o problema MAX-QBF com triplas proibidas, proposto na Atividade 8 de MO824 2S-2020.

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. Linearização com Triplas Proibidas

Para a linearização do problema MAX-QBF com triplas proibidas, utilizamos a seguinte formulação:

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

$$\text{s.a} \quad x_i + x_j + x_k \leq 2, \qquad \forall (i, j, k) \in T \quad (1.4)$$

$$w_{ij} \leq x_i \qquad \forall i \in \{1, ..., n\}, \forall j = \{1, ..., n\} \quad (1.5)$$

$$w_{ij} \leq x_j \qquad \forall i \in \{1, ..., n\}, \forall j =]\{1, ..., n\} \quad (1.6)$$

$$w_{ij} \geq x_i + x_j - 1 \qquad \forall i \in \{1, ..., n\}, \forall j = \{1, ..., n\} \quad (1.7)$$

$$x_i \in \mathbb{B}. \qquad \forall i \in \{1, ..., n\} \quad (1.8)$$

$$w_{ij} \geq 0 \qquad \forall i \in \{1, ..., n\}, \forall j = \{1, ..., n\} \quad (1.9)$$

1.3. 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

Para a avaliação, foram escolhidas versões de três metaheurísticas diferentes: GRASP, Busca Tabu e Algoritmos Genéticos, assim como implementações dos dois modelos apresentados na seção anterior. As implementações das metaheurísticas foram escolhidas por meio da qualidade dos resultados obtidos em seus respectivos testes.

2.1. GRASP

O GRASP é uma metaheurística que possui duas fases por iteração, sendo elas a construção e a busca local. A fase de construção, juntamente com a função que faz o reparo da solução, é responsável por gerar uma solução viável para o problema, por meio de uma heurística gulosa aleatorizada. Já a fase de busca local consiste em, a partir desta solução, varrer a vizinhança em busca de uma solução melhor. Por último, a melhor solução encontrada é atualizada, se necessário. Como critério de parada, o GRASP possui um número máximo de execuções e tempo limite.

O método de construção utilizado foi o Cost Perturbations, no qual a ideia principal é introduzir algum ruído nos custos originais. Essa abordagem incorpora mecanismos de aprendizagem associados a estratégias de intensificação e diversificação. O hiperparâmetro foi fixado em $\alpha = 0.4$. Note que esse método elimina a aleatoriedade da heurística construtiva.

2.2. Busca Tabu

A Busca Tabu é uma meta-heurística baseada em busca local que surgiu da ideia de utilizar uma **memória** para superar ótimos locais. A técnica permite a realização de movimentos que não melhoram a solução atual, evitando ciclos por meio de memórias chamadas de **listas tabu**. Uma lista tabu é uma estrutura de dados responsável por armazenar os últimos movimentos aplicados por um determinado número de iterações, a fim de evitar que eles sejam desfeitos por novos movimentos. Assim, uma descrição básica do método consiste em construir uma solução inicial e realizar iterativamente um movimento de vizinhança que não seja proibido, guardando-o na lista tabu.

O algoritmo básico consiste da construção de uma solução inicial (gulosa), seguida de uma série de movimentos de vizinhança, atualizando a melhor solução a

cada iteração e adicionando cada movimento na lista tabu (removendo a entrada mais antiga, caso necessário).

Além do procedimento básico, foram usadas estratégias de **intensificação** e **diversificação** para melhorar os resultados obtidos. Para diversificação foi utilizado *strategic oscillation*, método no qual a busca permite soluções inviáveis, aplicando uma penalidade a elas para estimular o retorno ao espaço viável. Também foi usada *intensificação por reinício*, na qual a cada i iterações sem melhora na solução titular, a busca é reiniciada a partir da melhor solução encontrada até então, com componentes fixas por j iterações.

Os hiperparâmetros foram fixados, com um *tenure* de 30 e intensificador (por reinício) com $i = 1000$ e $j = 100$. Na instância de tamanho 200, o parâmetro i é trocado para 2000.

2.3. Algoritmos Genéticos

Os Algoritmos Genéticos consistem em algoritmos de otimização baseados nos conceitos da teoria neodarwiniana da evolução. De forma geral, o funcionamento consiste na utilização de aleatoriedade na etapa de recombinação juntamente com a seleção da população segundo critérios de factibilidade e aptidão de indivíduos.

O algoritmo começa inicializando a população de cromossomos e em seguida realiza uma série de iterações correspondentes aos ciclos geracionais. Nestes ciclos são realizadas as seguintes etapas: seleção dos pais, recombinação (crossover), mutação, atualização da população e atualização da melhor solução. Tais passos representam as etapas fundamentais de um algoritmo genético. Para manter a viabilidade dos resultados, antes da atualização da população alguns genes são modificados em um processo de viabilização, para remover violações de triplas proibidas.

Como estratégias evolutivas alternativas foram utilizados *steady-state* e manutenção de diversidade. A estratégia *steady-state* consiste na escolha dos melhores cromossomos dentre pais e filhos para compor a nova população, de forma a manter o mesmo tamanho da população anterior e priorizar qualidade em possível detrimento de diversidade. Já a técnica para a manutenção dessa diversidade foca na escolha mais cautelosa dos *crosspoints* a serem usados no processo de *crossover*, de forma a evitar a geração de filhos idênticos.

Os hiperparâmetros foram fixados, com uma população de tamanho 50 e uma taxa de mutação dependente da instância $1/n$, onde n é o tamanho da instância.

3. Instâncias e Condições de Execução

A implementação das soluções heurísticas foi feita em *Java*, com base nos *frameworks* fornecidos, com as modificações citadas na Seção 2 e outras necessárias para obtenção dos dados e realização dos experimentos. Foi utilizado como *software* de resolução dos modelos matemáticos o *solver* Gurobi Optimizer V8.1.0, com a API para *Java*.

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.

O *hardware* usado para testes possui um processador Intel Core i5 Quad-Core, com 2, 3 GHz de frequência do *clock*. O computador possui 8 GB de RAM, operando com 2133 MHz. O sistema operacional utilizado foi o macOS Catalina versão 10.15.6 (64 bits).

4. TTT-Plots

O método de *TTT-Plots* para avaliação de desempenho foi formalizado por Aiex, Resende e Ribeiro em 2005 [1], mesmo já sendo usado anteriormente. Gráficos *time-to-target* (TTT) mostram a probabilidade de um algoritmo encontrar uma solução ao menos tão boa quanto um determinado valor alvo k no tempo (de CPU).

Seguindo a hipótese de que os tempos de CPU seguem uma distribuição exponencial deslocada, podemos executar um método heurístico n vezes em uma instância fixa, marcando o tempo necessário para encontrar uma solução com valor maior ou igual à referência k em cada. Em cada execução o gerador de números pseudo-aleatórios é inicializado com uma *seed* diferente.

Para a construção do gráfico em uma instância, os tempos são ordenados crescentemente, e uma probabilidade $p_i = (i - 1/2)/n$ é associada a cada tempo t_i. Esses valores são então relacionados a cada tempo no gráfico.

Dado o gráfico, pela hipótese, podemos estimar os parâmetros da distribuição equivalente. Como a função de distribuição cumulativa é $F(t) = 1 - e^{-(t-\mu)/\lambda}$, queremos encontrar os parâmetros λ (média) e μ (deslocamento). Para isso, é possível utilizar outro gráfico, chamado *Q-Q-plot*.

Para cada p_i, temos um **quantil** $Qt(p_i)$ da distribuição teórica, tal que $F(Qt(p_i)) = p_i$. Assim, $Qt(p_i) = F^{-1}(p_i) = -\lambda \ln(1 - p_i) + \mu$. Na distribuição empírica, os quantis são os dados ordenados. Assim, um *Q-Q-plot* é obtido ao relacionar os quantis teóricos com os empíricos em um gráfico. Em um caso no qual a distribuição teórica é uma aproximação da empírica, esse gráfico forma algo similar a uma reta, dada por $y = \hat{\lambda}x + \hat{\mu}$. Os parâmetros λ e μ podem ser estimados por $\hat{\lambda}$ e $\hat{\mu}$, respectivamente.

Para encontrar essas estimativas, evitando distorções por *outliers*, o gradiente da reta é calculado por meio dos quartis inferior ($q_l = Q(1/4)$) e superior ($q_u = Q(3/4)$): $\hat{\lambda} = [t_u - t_l]/[q_u - q_l]$, sendo os tempos t_i ordenados. Consequentemente, o deslocamento é estimado por $\hat{\mu} = z_l - \hat{\lambda}q_l$.

A reta dada pelos parâmetros estimados pode então ser sobreposta no gráfico Q-Q, e para cada ponto o intervalo de um desvio padrão σ é também sobreposto, sendo uma estimativa $\hat{\sigma} = \hat{\lambda}[p_i/(1 - p_i)n]^{\frac{1}{2}}$. Finalmente, com os parâmetros estimados pode-se construir um gráfico com a sobreposição das distribuições teórica e empírica.

4.1. Implementação, Configurações e Parâmetros

O *script* para construção dos gráficos TTT usado foi implementado em *Perl* por Aiex, Resende e Ribeiro [1] (`tttplots.pl`). Nele, são produzidos o gráfico Q-Q com informação de variabilidade sobreposta e o gráfico com sobreposição das distribuições teórica e empírica. A entrada do *script* é um arquivo `input.dat` com n tempos de CPU, um por linha. Requer o programa externo `gnuplot`. A versão mais atual do `gnuplot` (5.4.1, no momento do teste) não identifica o arquivo `gpl` gerado pelo *script*, então uma versão mais antiga é necessária. A versão usada foi a 4.0.0, de 2004.

A escolha de n é muito importante para os gráficos. Quanto maior o valor de n, melhor a relação entre as distribuições teórica e empírica, porém mais tempo é necessário para execução. Para balancear os recursos limitantes (tempo e disponibilidade), foi escolhido $n = 100$ para uma única instância, de tamanho 100.

Para a instância, foram usados dois valores alvo k, com diferentes níveis de dificuldade. O valor mais fácil foi $k_1 = 1100$, e o mais difícil foi $k_2 = 1200$.

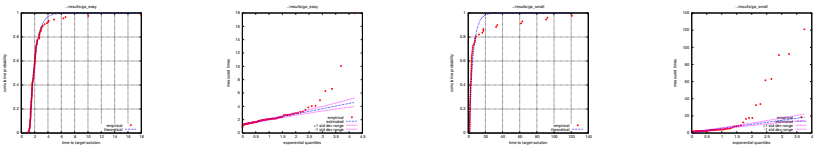
Esses valores foram escolhidos após testes empíricos com valores diferentes: os valores iniciais eram $k_1 = 1000$ e $k_2 = 1263$, porém se mostraram inadequados. O valor mais baixo foi muito fácil de ser atingido, sendo superado já na primeira solução encontrada por dois dos métodos testados, enquanto o valor mais alto foi muito difícil de ser atingido, com alta taxa de falhas. Por isso, os gráficos não seguiam nenhum tipo de distribuição exponencial, e nenhuma análise pôde ser feita, e assim os valores intermediários foram escolhidos.

Os métodos foram executados com limite de 5000 iterações/gerações e tempo máximo de 600 segundos (10 minutos), sem limite de memória. A mesma sequência de *seeds* foi usada para todos os métodos, os n primeiros números de um gerador pseudoaleatório com *seed* 1327.

4.2. Resultados e Análise

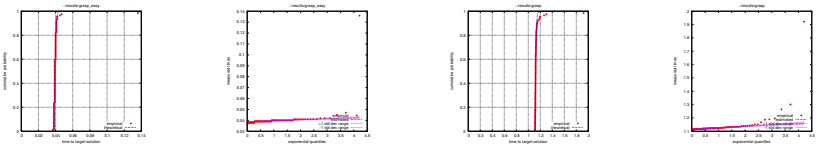
Os resultados para o GRASP, a Busca Tabu e o Algoritmo Genético podem ser vistos nas Figuras 1, 2 e 3, respectivamente. Para o Algoritmo Genético, os gráficos para o alvo difícil foram feitos após remoção das execuções nas quais o alvo não foi atingido. Isso ocorreu em 37 das 100 vezes, então esses gráficos possuem apenas 63 pontos. Pelos gráficos resultantes, pode-se observar diferentes níveis de adequação dos métodos com as distribuições exponenciais teóricas.

O método que melhor entrou nos padrões teóricos foi o Algoritmo Genético (Figura 1), principalmente ao se considerar o alvo fácil. Ainda assim, ainda há uma grande quantidade de *outliers*, então os dados não encaixaram perfeitamente na distribuição. Para o alvo fácil, menos de 10% dos dados coletados ultrapassaram a linha de um desvio padrão (com poucos desses se afastando muito dessa linha), o que pode ser causado pelo ambiente de teste (processo comum em um sistema operacional complexo com serviços e alarmes periódicos) ou por *seeds* que geraram uma sequência particularmente ruim de escolhas de reprodução/mutação. Isso pode ser ainda mais visível graças à forma de viabilização descrita na Seção 2.3, que



(a) TTT Plot (Fácil) (b) Q-Q Plot (Fácil) (c) TTT Plot (Difícil) (d) Q-Q Plot (Difícil)

Figura 1: Gráficos TTT e Q-Q para Algoritmo Genético



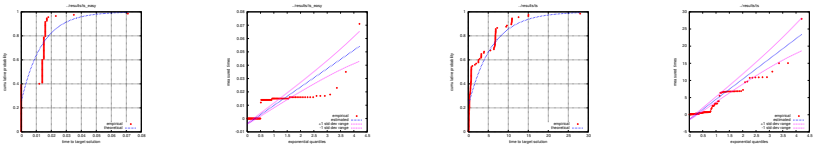
(a) TTT Plot (Fácil) (b) Q-Q Plot (Fácil) (c) TTT Plot (Difícil) (d) Q-Q Plot (Difícil)

Figura 2: Gráficos TTT e Q-Q para GRASP

pode negar mudanças feitas na reprodução/mutação. Para o alvo difícil, a taxa de *outliers* é maior, colocando em dúvida a compatibilidade da distribuição. Por fim, nota-se que o desvio padrão é pequeno, então os valores que não se destacaram se mantiveram próximos.

O GRASP (Figura 2), como descrito na Seção 2.1, não possui aleatoriedade, então qualquer variação é graças ao ambiente de execução (os passos são exatamente iguais). O resultado é uma linha quase vertical, com um crescimento muito rápido, pois os tempos foram quase todos iguais, com alguns *outliers*. Por isso, os gráficos Q-Q são extremamente estreitos, quase sem desvio padrão. Essa característica de execução também é visível pela semelhança entre os gráficos para ambos os alvos. Enfim, esses gráficos não são adequados para caracterização desse método.

Por fim, a Busca Tabu (Figura 3) é o caso com comportamento mais diferente dos outros. Para o alvo difícil, os tempos seguem aproximadamente a distribuição teórica, mas basta olhar para o gráfico Q-Q para notar que o método não segue uma distribuição exponencial deslocada. Os resultados para o alvo fácil são ainda



(a) TTT Plot (Fácil) (b) Q-Q Plot (Fácil) (c) TTT Plot (Difícil) (d) Q-Q Plot (Difícil)

Figura 3: Gráficos TTT e Q-Q para Busca Tabu

menos compatíveis. Em vez de uma reta, a figura formada se assemelha mais a uma **escada**, que pode ser vista nos quatro gráficos.

Comparando os três métodos pelos gráficos, percebe-se que, para essa instância, o GRASP produz soluções com custo maior que o alvo difícil com um tempo em média menor que os outros métodos, com a vantagem de um desvio padrão pequeno, por ser determinístico. Já o alvo fácil é alcançado mais rapidamente pela Busca Tabu, o que pode indicar uma solução inicial melhor. De fato, a Busca Tabu possui solução inicial gulosa, enquanto o GRASP seleciona os elementos aleatoriamente de uma lista restrita de candidatos. O Algoritmo Genético é claramente o mais lento para convergir, e alcançar qualquer solução boa.

5. Perfis de Desempenho

Considere que um conjunto P de n_p problemas seja utilizado no teste de um conjunto S de n_s métodos de solução. Para cada problema $p \in P$ e método de solução $s \in S$, seja $t_{p,s}$ o tempo computacional necessário para a resolução de p utilizando s. Caso uma outra métrica seja utilizada, basta definir $t_{p,s}$ de forma adequada. O desempenho do método de solução $s \in S$ na resolução de $p \in P$ é analisado pelo *raio de desempenho*, dado por $r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s}:\forall\ s\in S\}}$.

Observe que esta razão é sempre maior ou igual a 1. Se o método de solução s falhar na resolução do problema p então $r_{p,s}$ é definido como r_M, tal que $r_M \geq r_{\overline{p},\overline{s}}, \forall\ \overline{p} \in P, \forall\ \overline{s} \in S$.

Neste trabalho, considerou-se $r_M = \max\{r_{\overline{p},\overline{s}} : \forall\ \overline{p} \in P, \forall\ \overline{s} \in S\}$. Conforme mostrado em [2], a escolha de r_M não influencia na avaliação do desempenho.

A razão de desempenho mostra o comportamento de um método de solução na resolução de um determinado problema. Entretanto, o que se deseja é uma avaliação geral do desempenho do método de solução. Para isso, é definido o *perfil de desempenho*, dado por: $\rho_s(\tau) = \frac{1}{n_p}|\{p \in P : \log(r_{p,s}) \leq \tau\}|$, com $|.|$ representando o número de elementos no conjunto. O perfil de desempenho ρ_s é uma função que associada um dado valor $\tau \in \mathbb{R}$.

Um perfil de desempenho também pode ser visto como a probabilidade de que a razão de desempenho $r_{p,s}$ associada ao método de solução s esteja dentro de um fator τ da melhor razão obtida. Com isso, $\rho_s(\tau)$ corresponde a uma função de distribuição acumulada para a razão de desempenho, dada por: $\rho_s(\tau) = Pr(r_{p,s} \leq \tau : 1 \leq s \leq n_s)$.

Pode-se dizer também que $\rho_s(\tau)$ é a probabilidade de que o método de solução s resolva um problema de P em não mais do que τ vezes o tempo mínimo levado por qualquer outro método de solução em S. Se o conjunto P é suficientemente grande e capaz de representar os problemas provavelmente encontrados na prática, então os métodos de solução com grande probabilidade $\rho_s(\tau)$ são preferidos.

Para analisar um perfil de desempenho, alguns pontos principais devem ser observados. O primeiro deles é o valor associado a $\tau = 1$. Para um dado método de solução $s \in S$, o valor $\rho_s(\tau = 1)$ corresponde a fração de problemas para os quais s obteve o melhor desempenho que os demais métodos de solução. Além disso, é

importante verificar os casos onde $\rho_s(r_M) = 1$ já que $r_{p,s} = r_M$ quando o problema p não é resolvido pelo método de solução s.

Assim como demostrado por Dolan e Moré [2], os perfis de desempenho não são sensíveis aos resultados em um pequeno número de problemas, pois o desempenho obtido em um problema particular do conjunto não afeta de modo significativo os perfis de desempenho associados. Além disso, os perfis são dificilmente afetados por pequenas mudanças nos resultados associados a muitos problemas.

5.1. Implementação, Configurações e Parâmetros

Para a geração dos gráficos de Performance de Desempenho, foi utilizado como base o *script* (`perf.m`), em MATLAB, disponibilizado por E.D. Dolan and J.J. More [2]. Graças à característica de maximização do problema, o *script* foi reescrito em Python (`pp.py`) com as alterações necessárias. Assim, foi possível gerar o gráfico a partir de uma tabela de entrada contendo em cada coluna o método de resolução e em cada linha o tempo de execução ou o custo de solução de cada instância. Para a criação do gráfico foram utilizadas bibliotecas auxiliares, dentre elas `NumPy`, `Pandas` e `MatPlotLib`.

A tabela de entrada do programa foi construída com os resultados da execução de todos os métodos implementados para o problema, sendo eles: GRASP, Busca Tabu, AG, modelo inteiro quadrático para o MAX-QBFPT e modelo linear inteiro misto para o MAX-QBFPT, para todos os 7 tamanhos de instâncias, $20, 40, 60, 80, 100, 200, 400$.

Os métodos foram executados com número máximo de 10000 iterações e tempo máximo de 1800 segundos (30 minutos), sem limite de memória.

5.2. Resultados e Análise

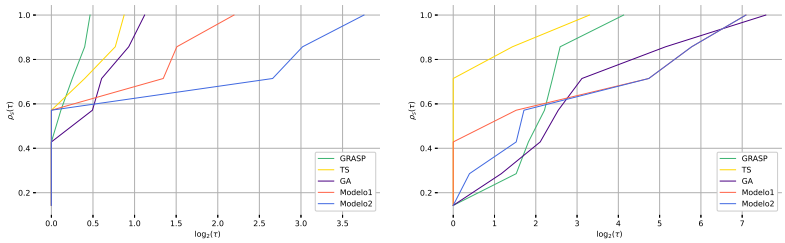
A Tabela 1 resume os resultados obtidos pelos cinco métodos de solução descritos anteriormente. As colunas mostram em ordem o valor da função objetivo (*Solução*) e tempo de execução (*Tempo*). Especificamente, para o Modelo1 e Modelo2, a coluna *Solução* mostra em ordem os valores da função objetivo e do limitante superior.

Tabela 1: Resultados obtidos para cada método de solução.

	GA		GRASP		TS		Modelo1		Modelo2	
Instância	Solução	Tempo (s)	Solução	Tempo (s)	Solução	Tempo (s)	Solução	Tempo (s)	Solução	Tempo (s)
qbf020	125	15.8	125	1.5	120	0.8	**125**	0.1	**125**	0.1
qbf040	366	57.0	366	5.7	366	4.4	**366**	1.6	**366**	5.3
qbf060	495	114.6	500	79.5	500	13.2	[508, 868]	1800.0	[508, 580]	1800.0
qbf080	813	191.6	843	152.0	851	32.8	[851, 1945]	1800.0	[851, 1327]	1800.0
qbf100	1189	290.8	1263	357.5	1263	67.5	[1166, 3336]	1800.0	[928, 2282]	1800.0
qbf200	4038	1402.2	4105	1800.3	4122	625.2	[3385, 21939]	1800.0	[2658, 14071]	1800.1
qbf400	11792	1800.6	11568	1801.2	11378	1800.4	[10706, 73019]	1800.1	[4418, 75512]	1800.1

As Figuras 4a e 4b mostram os perfis de desempenho obtidos para os cinco métodos de solução referenciados como GRASP, TS, GA, Modelo1 e Modelo2. Lembrando que para a construção desses gráficos, um mesmo conjunto de problemas

foi resolvido por cada um dos métodos de solução no qual os valores da função objetivo e tempos de execução associados foram coletados. Como o problema aqui estudado é de maximização, para cada função objetivo, o valor de $t_{p,s}$ é dado por $t_{p,s} = 1,05 \cdot \max\{f_{p,s} | s \in S\} - f_{p,s}$, onde $f_{p,s}$ é o valor da função objetivo obtido pelo método de solução $s \in S$ aplicado ao problema $p \in P$. A multiplicação por $1,05$ tem o propósito de evitar divisão por zero. A partir desses dados, foi calculado $r_{p,s}$ e então os perfis de desempenho.



(a) Função objetivo (b) Tempo de execução

Figura 4: Perfis de desempenho

Os perfis de desempenho na Figura 4a comparam os cinco métodos de solução em termos do valor da função objetivo. Observe que o GRASP, TS, Modelo1 e Modelo2 obtiveram o melhor desempenho para quase 58% dos problemas, enquanto que o GRASP e TS obtiveram o melhor desempenho para cerca de 43% dos problemas. Note que o Modelo1 se destaca em relação ao Modelo2, dado que a curva de seu perfil de desempenho do Modelo1 domina a do Modelo2. Isso não indica que o Modelo1 obteve o melhor desempenho que o Modelo2 e, sim, que o Modelo1 conseguiu resolver mais problemas dentro de um fator τ de desempenho do que o Modelo2.

Já na Figura 4b os métodos de solução são comparados em termos do tempo de execução. A TS tem o melhor desempenho para quase 70% dos problemas, o que significa que a TS é capaz de resolver 70% dos problemas tão rápido ou mais rápido que as demais abordagens. O Modelo1 resolve cerca de 43% dos problemas tão rápido ou mais rápido do que as demais abordagens. Note que todos os métodos de solução foram capazes de resolver todos os problemas, dado que a curva de seus perfis de desempenho atingiram valor 1 para $\tau = [1, r_M)$.

Combinando os perfis de desempenho com a Tabela 1, pode-se ver que, embora todos os métodos tenham seu valor, resolvendo bem os problemas, a Busca Tabu (ou TS) se destaca por conseguir ótimos resultados de modo eficiente, gastando menos tempo que os outros métodos.

Referências

[1] R. M. Aiex, M. G. C. Resende, and C. C. Ribeiro, "Ttt plots: a perl program to create time-to-target plots," *Optimization Letters*, vol. 1, pp. 355–366, 2007.

[2] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical Programming*, vol. 91, no. 2, pp. 201–213, 2002.