MO824 - Análise Comparativa entre as metaheurísticas *GRASP*, *Tabu*, e *GA* para a resolução do problema MAX-QBF com mochila

Lucas Guesser Targino da Silva (203534)

4 de julho de 2022

Esse trabalho tem como objetivo comparar os resultados obtidos pelas implementações de três metaheurísticas:

1. *Greedy Randomized Adaptive Search Procedure* (*GRASP*) [1]
2. *Tabu Search* (*Tabu*) [2]
3. *GA Algorithm* (*GA*) [3]

O problema resolvido foi o *MAX-QBF com mochila*, descrito no Apêndice A.

1 Metaheurísticas

Nessa seção são descritas todas as metaheurísticas utilizadas ao longo do trabalho.

No desenvolvimento dos trabalhos anteriores, explorou-se variações de cada metaheurística, essas com diferentes parâmetros, estratégias construtivas, buscas locais, etc. Para cada metaheurística, foram selecionadas duas variações, as que apresentaram melhor desempenho, para serem utilizadas nas investigações deste trabalho.

1.1 *GRASP*

A metaheurística *GRASP* é descrita no Algoritmo 1, sua estratégia construtiva em Algoritmo 2, e sua estratégia de busca local em Algoritmo 3, todos descritos no Apêndice A.2.

Ambas usam a estratégia construtiva padrão (Algoritmo 2) e o parâmetro α com valor 0.2.

Elas diferem na estratégia de busca local, entretanto. A primeira variação, chamada *GRASP Best*, utiliza *Best Improving*, em que toda a vizinhança é percorrida e a melhor opção selecionada. A segunda variação, chamada *GRASP First*, utiliza *First Improving*, em que a busca na vizinhança retorna a primeira solução encontrada que seja melhor do que a atual.

Em ambas, *Best Improvinge First Improving*, a vizinhança é definida como o conjunto de soluções obtidas a partir da solução atual que tenham:

1. 1 elemento a mais - adição;
2. 1 elemento a menos - remoção;
3. 1 elemento trocado (equivalente a uma adição e uma remoção) - troca;

1.1.1 *GRASP Best*

1. Estratégia construtiva: **padrão** com α igual a **0.2**
2. Estratégia de busca local: **best improving**

1.1.2 *GRASP First*

1. Estratégia construtiva: **padrão** com α igual a **0.2**
2. Estratégia de busca local: **first improving**

1.2 *Tabu*

A metaheurística *Tabu* é descrita no Algoritmo 4.

Ambas as variações descritas nessa subseção usam:

1. solução inicial a solução da estratégia construtiva do *GRASP*, Algoritmo 2;
2. busca local *Best Improving*. Ela é similar a *Best Improving*descrita na Subseção 1.1, entretanto ela não considera a adição ou remoção de elementos que estão na lista tabu T (a menos que eles levem a uma solução melhor do que S^*);
3. *Tenure Ration*, parâmetro que controla o tamanho da lista tabu T, igual a 0.4, o que significa que T pode ter tamanho até 40% do tamanho da entrada.

A primeira variação, chamada *Tabu Vanilla*, implementa *Tabu* com as características acima. A segunda variação, chamada *Tabu com Intensificação e Diversificação*, inclui estratégias de diversificação e intensificação, descritas nas Subsubseção 1.2.1 e Subsubseção 1.2.2.

1.2.1 Estratégia de Intensificação

A estratégia de intensificação aumenta o tamanho da vizinhaça, ao invés de considerar 1 adição, 1 remoção e 1 troca, ela considera:

1. 2 adições e 1 remoção;
2. 2 remoções e 1 adição;
3. 2 adições e 2 remoções;

A intensificação é feita em torno da melhor solução conhecida. Ela é ativada quando passam-se muitas iterações (o critério de parada é definido como um número máximo de iterações, e "muitas iterações" significa 20% número máximo de iterações) sem melhora na solução ótima e sem sua ativação.

1.2.2 Estratégia de Diversificação

A estratégia de diversificação constroi uma nova solução, utilizando a estratégia construtiva do GRASP, e recomeça a busca nesse outro local do espaço de solução. Além disso, antes de começar a busca, faz-se uma busca intensiva (usando a estratégia de intensificação) em torno dessa nova solução.

Seu critério de ativação é o mesmo da Estratégia de Intensificação, excetuando-se que Diversificação é ativada apenas quando Intensificação não é (no programa, há um registro separado para quando cada uma delas foi ativada pela última vez).

Assim, quando se passaram muitas iterações sem melhora na solução ótima, primeiro tenta-se intensificação. Caso essa falhe, executa-se a diversificação.

1.2.3 *Tabu Vanilla*

1. Solução inicial: saída da estratégia construtiva do *GRASP*
2. *Tenure Ration*: **0.4**
3. Estratégia de busca local: ***Best Improving***

1.2.4 *Tabu com Intensificação e Diversificação*

1. Solução inicial: saída da estratégia construtiva do *GRASP*
2. *Tenure Ration*: **0.4**
3. Estratégia de busca local: ***Best Improving***
4. Adição das estratégias: **Intensificação e Diversificação**

1.3 *GA*

A metaheurística *Tabu* é descrita no Algoritmo 5.

Ambas as variações descritas nessa subseção usam:

1. população inicial aleatória;
2. tamanho da população igual a 100;
3. seleção para reprodução em torneio: dois cromossomos são escolhidos aleatoriamente e o melhor dos dois (em relação à função de aptidão) é escolhido para continuar enquanto o outro é descartado.
4. reprodução com *two point crossover (2X)*;
5. Critério de parada: 1000 gerações;

1.3.1 *GA Vanilla*

A primeira variação, chamada *GA Vanilla*, implementa *GA* conforme descrito na Subseção 1.3 com as seguintes adições/modificações:

1. taxa de mutação igual a 0.5%;
2. seleção da nova população: descendentes, substituindo-se o pior deles pelo melhor gene conhecido

1.3.2 *GA Steady*

A segunda variação, chamada *GA Steady*, implementa *GA* conforme descrito na Subseção 1.3 com as seguintes adições/modificações:

1. taxa de mutação igual a 1%;
2. seleção da nova população: os 100 melhores genes entre mães e filhos (conhecida como *Steady-State* $\lambda + \mu$);

2 Time-To-Target Plot (TTT Plot)

2.1 Entendedo o TTT Plot

Considere um problema p, um algoritmo de metaheurística A, e um valor alvo v. Seja t a variável randomica que representa o tempo que A leva para encontrar uma solução com valor pelo menos v[1]:

$$t = \text{TempoExecução}(A(p) \geqslant v) \tag{1}$$

t é chamada *Tempo para atingir valor alvo da solução*[2].

Em [4], os autores conjecturam que a distribuição de t é uma função exponencial deslocada:

$$P(t) = exp\left(\frac{-(t-\mu)}{\lambda}\right) \quad \lambda \in \mathbb{R}^+, \mu \in \mathbb{R} \tag{2}$$

Sem TTT Plot, como compararíamos o desempenho dos algoritmos de metaheurística? Possivelmente com tabelas com alguns resultados e alguma análise estatística menos desenvolvida.

O TTT Plot fornece um modelo para o comportamento esperado dos algoritmos. Isso permite que comparemos os seus desempenhos de forma mais robusta já que ele requer que:

1. juntemos informação suficiente para criar o modelo;
2. criemos o modelo;

Além do modelo exponencial, o TTT Plot inclui um gráfico *Q-Q plot*, que permite analisarmos a validade do modelo exponencial para um certo conjunto de dados.

2.2 Escolha dos Problemas e Valores Alvo

Para a análise desse problema, escolheu-se as instâncias do MAX-KQBF e valores alvo da Tabela 1 abaixo (a nomeclatura dos problemas segue a Tabela 4).

Problema	**Valor Alvo**
kqbf040	275
kqbf060	446
kqbf080	729

Tabela 1: Instâncias do MAX-KQBF selecionadas e seus respectivos valores alvo.

O valor alvo de cada instância foi selecionado como o limite inferior do intervalo de otimalidade da Tabela 4. Tais valores foram escolhidos por não serem baixos, o que comprometeria a análise [4], mas também não são altos demais, o que comprometeria o tempo de execução.

Selecionou-se as instâncias *kqbf040*, *kqbf060*, *kqbf080* pois elas apresentam o grau de dificuldade requerido mas não requerem tanto tempo de execução (o que inviabilizaria a execução dos testes já que o tempo requerido seria muito grande).

2.3 Validação dos TTT Plot

Os TTT Plot para todos os algoritmos analisados nesse trabalho estão no Apêndice C.

[1] Aqui supõe-se um problema de maximização, mas funciona de forma similar para minimização.

[2] *Time-To-Target Solution Value* em inglês.

2.3.1 *GRASP Best*

Figuras: 3, 9, 15.

Observando os *Q-Q plot* nas Figuras 9, 15, vemos que o modelo proposto descreve bem o comportamento do algoritmo.

Porém, o modelo da Figura 3 não. Na verdade, observamos dois conjuntos bem claros de tempo de execução: os próximos de zero e os em torno de 3.5 segundos. Esses dois conjuntos correspondem aos casos em que o algoritmo atingiu e os casos em que ele não atingiu o valor alvo, respectivamente[3]. Para os últimos, o tempo de execução tende a ser constante, que é o tempo necessário para a execução de todas as iterações.

Estranhamente, foi observado que ou o algoritmo encontra uma solução nas primeiras iterações (e converge rapidamente), ou ele não encontra uma solução satisfatória em nenhuma iteração. Isso é de certa forma conflitante com o reinicio da solução a cada iteração e mereceria mais investigações. Esse não é, entretanto, o objetivo da presente atividade.

2.3.2 *GRASP First*

Figuras: 4, 10, 16. Observando-as, vemos que o modelo proposto descreve bem o comportamento do algoritmo.

2.3.3 *Tabu Vanilla*

Figuras: 5, 11, 17. Observando-as, vemos que nenhum modelo proposto descreve bem o comportamento do algoritmo.

Para os casos da Figura 5, observando os logs de execução, vê-se dois comportamentos bem distintos:

1. execuções em que o valor alvo é atingido em poucas iterações (menos de 10). Tempo de execução em torno de alguns poucos milisegundos ($\approx 10ms$).

2. execuções em que são necessárias pelo menos 300 iterações para atingir o valor alvo. Tempo de execução em torno de décimos de segundo ($\approx 400ms = 0.4s$).

Mas isso não é suficiente para explicar todos os comportamentos observados. Houveram casos em que, mesmo com poucas iterações, o tempo de execução foi muito grande, como é em alguns o caso das execuções da Figura 11. Para esses, não encontrou-se nenhuma explicação.

2.3.4 *Tabu com Intensificação e Diversificação*

Figuras: 6, 12, 18. Observando-as, vemos que nenhum modelo proposto descreve bem o comportamento do algoritmo. O comportamento observado é similar ao descrito na Subsubseção 2.3.3.

2.3.5 *GA Vanilla*

Figuras: 2, 8, 14. Observando-as, vemos que, à exceção de alguns ponto fora da curva, o modelo proposto descreve bem o comportamento do algoritmo.

[3] Observando-se os logs isso fica bem claro.

2.3.6 *GA Steady*

Figuras: 1, 7, 13. Observando-as, notamos que o modelo não parece ser tão adequado para essas instâncias. Nos gráficos de Distribuição de Probabilidade Cumulativa, parece haver uma região de plato, em que um aumento no tempo computacional não leva a uma maior probabilidade de encontrar uma solução boa.

Observou-se que a variação de tempo de execução deve-se ao número de iterações necessárias para a convergência do algoritmo: enquanto que alguns casos precisam de algumas poucas dezenas de iterações, outros precisam de algumas centenas. Isso pode ser explicado por uma população inicial ruim, ou talvez perda de diversidade. A verificação de que é algum desses fatores, ou nenhum deles, que causa o comportamente esperado está, entretanto, fora do escopo do presente trabalho.

2.4 Comparação das metaheurísticas usando TTT Plot

Metaheurística	**Instância**	**Tempo 90%** [s]
GA Vanilla	kqbf040	1.50
GA Steady	kqbf040	11.00
GRASP First	kqbf040	0.09
GRASP Best	kqbf040	4.00
Tabu Vanilla	kqbf040	0.65
Tabu com Intensificação e Diversificação	kqbf040	0.60
GA Vanilla	kqbf060	3.00
GA Steady	kqbf060	37.00
GRASP First	kqbf060	0.40
GRASP Best	kqbf060	0.08
Tabu Vanilla	kqbf060	5.00
Tabu com Intensificação e Diversificação	kqbf060	3.50
GA Vanilla	kqbf080	7.50
GA Steady	kqbf080	75.00
GRASP First	kqbf080	0.90
GRASP Best	kqbf080	0.25
Tabu Vanilla	kqbf080	10.00
Tabu com Intensificação e Diversificação	kqbf080	7.00

Tabela 2: Tempo requerido por cada metaheurística para atingir, com 90% de chance, o valor alvo. Dados extraídos por inspeção visual dos gráficos do Apêndice C.

Observa-se na tabela 2 que as metaheurísticas que apresentam os melhores resultados são *GRASP Best* e *GRASP First*. *GRASP Best* possui um comportamento diferente na instância kqbf040 (reveja a Figura 3). Se não fosse isso, poderia-se dizer que essa é a variação com melhor desempenho.

GA Vanilla e *Tabu com Intensificação e Diversificação* possuem desempenhos praticamente equivalentes. Já *Tabu Vanilla* está um pouco atrás dessas, o que mostra que as estratégias alternativas implementadas afetam positivamente.

GA Steady apresentou desempenho pior do que todas as outras metaheurísticas em todos os casos observados, pior especialmente do que a variação *GA Vanilla*. Isso mostra que, para esse problema, as modificações implementadas não são vantajosas.

3 Performance Profiles

3.1 Motivação

Em geral, quando analisamos algoritmos, estamos interessados em determinar aquele que melhor resolve o problema em questão. Para isso, são feitos o que chamamos de *benchmarks*: análises comparativas do desempenho vários algoritmos. Tais comparações envolvem:

1. medições de desempenho: tempo de CPU, número de iterações, número de chamadas de função, etc;
2. comparação de resultados: definir um valor alvo e verificar se o algoritmo consegue atingi-lo/encontrá-lo;

Compilar todos esses resultados de forma objetiva, útil, e fácil de visualizar, não é simples. Problemas comuns encontrados são apontados em [5]:

1. não reportar falhas;
2. resultados dominados por instâncias difíceis;
3. difícil visualização e entendimento dos resultados;
4. subjetividade na análise;

Para resolver tal problema, foi desenvolvido em [5] *Performance Profile*.

3.2 Descrição do *Performance Profile*

Dados:

1. conjunto de problemas $\mathcal{P}, n_p = |\mathcal{P}|$;
2. conjunto de algoritmos de resolução (solvers) $\mathcal{S}, n_s = |\mathcal{S}|$;

Definimos

1. $t_{p,s}$: tempo requerido para resolver o problema p pelo solver s;
2. $r_{p,s} = \dfrac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}}$: taxa de desempenho;
3. $\rho(\tau) = \dfrac{1}{n_p}\ size\{p \in \mathcal{P} : r_{p,s} \leqslant \tau\}$: função de distribuição cumulativa para a taxa de desempenho[4];

Observer que $\rho(\tau)$ é a probabilidade do solver s ter uma taxa de desempenho $r_{p,s}$ num fator τ. Além disso, $\rho(1)$ é a probabilidade do solver $s \in \mathcal{S}$ ter desempenho melhor do que os outros solvers $s' \in \mathcal{S} \setminus \{s\}$.

[4] *cummulative distribution function for the performance ratio.*

3.3 Confecção dos Gráficos *Performance Profile*

Para a confecção dos graficos de *Performance Profile*apresentados no Apêndice D, foram usados os dados dos trabalhos anteriores. Definiu-se como valor de referência o limite inferior do intervalo de otimalidade da Tabela 4. Tratando-se de uma análise de metaheurísticasm, foi escolhido tal valor por dois motivos:

1. não há garantias de encontrar o ótimo em nenhum dos método apresentados;
2. já que são resolvidos problemas difíceis, o objetivo é encontrar valores próximos do ótimo, de forma que estando dentro de um gap de otimalidade, a solução é considerada boa;

Dessa forma, os dados foram compilados na Tabela 5 e utilizados para a confecção dos gráficos Figura 19, Figura 20, e Figura 21.

3.4 Análise de Desempenho Usando *Performance Profile*

Figuras de *Performance Profile*são apresentadas no Apêndice D.

Nota-se na figura Figura 19 que o *GRASP First* tem a maior probabilidade de ser o melhor para um problema, com chance de aproximadamente 60%. Logo atrás estão *GRASP Best* e *GA Vanilla* com quase 20% e os outros com chance de 0%.

Considerando um fator de $\tau = 2^5$, a chance de algum *GRASP First* estar nesse fator é quase 90% e do *GRASP Best* é de aproximadamente 70%.

As metaheurísticas *Tabu Vanilla*, *Tabu com Intensificação e Diversificação*, *GA Vanilla*, e *GA Steady* apresentaram desempenho consideravelmente pior do que *GRASP Best*.

4 Melhores Soluções

Aviso: Qualquer anlálise da tolerância aceita, em relação à melhor solução, dever levar a aplicação em consideração. Para esse trabalho, não há uma aplicação específica, de forma que é difícil apontar uma tolerância aceitável. mas considerando que estão sendo analisadas heurísticas, para fins de análise, qualquer tolerância dentro de 5% será considerada aceitável.

[5]Isto é, a taxa de desempenho ser até 2, o algoritmo executar até 1.7 vezes o tempo do melhor algoritmo.

4.1 Resultados

Metaheurística	Instância	Melhor Solução	% do Melhor
GA Vanilla	kqbf040	303	1.62
GA Steady	kqbf040	305	0.97
GRASP First	kqbf040	306	0.64
GRASP Best	kqbf040	295	4.22
Tabu Vanilla	kqbf040	308	0.00
Tabu com Intensificação e Diversificação	kqbf040	303	1.62
GA Vanilla	kqbf060	481	2.03
GA Steady	kqbf060	453	7.73
GRASP First	kqbf060	455	7.33
GRASP Best	kqbf060	491	0.00
Tabu Vanilla	kqbf060	491	0.00
Tabu com Intensificação e Diversificação	kqbf060	491	0.00
GA Vanilla	kqbf080	766	3.64
GA Steady	kqbf080	781	1.76
GRASP First	kqbf080	783	1.50
GRASP Best	kqbf080	795	0.00
Tabu Vanilla	kqbf080	783	1.50
Tabu com Intensificação e Diversificação	kqbf080	702	11.69

Tabela 3: Melhor solução encontrada por cada metaheurística, para cada instância analisada. % do Melhor é calculada como $100 \cdot \left(1 - \frac{s}{s^*}\right)$, sendo s a solução da metaheurística em questão e s^* a melhor solução encontrada para aquela instância.

4.2 Análise

A Tabela 3 contém resultados com as melhores soluções encontradas por cada metaheurística para as instâncias selecionadas.

Para a instância kqbf040, todas as metaheurísticas encontraram soluções dentro de 5% da melhor.

Para a instância kqbf060, *GRASP Best*, *Tabu Vanilla* e *Tabu com Intensificação e Diversificação* encontraram a mesma solução, e *GA Vanilla* está dentro dos 5%. Já *GA Steady* e *GRASP Best* não encontraram uma solução satisfatória.

Para a instância kqbf080, a única que teve desempenho não satisfatório foi *Tabu com Intensificação e Diversificação*.

Em geral, as melhores metaheurísticas foram *Tabu Vanilla*, *GRASP Best*, *GA Vanilla*, nessa ordem. Elas forma as únicas a ficaram dentro da tolerância de 5% nas três instâncias. *Tabu Vanilla* e *GRASP Best* obtiveram duas vezes a melhor solução, sendo que a segunda foi melhor na média.

5 Conclusão

Nesse trabalho, foram apresentadas análises comparativas de seis metaheurísticas utilizando:

1. Melhores Soluções - Seção 4;
2. Time-To-Target Plot- Seção 2;

3. *Performance Profile*- Seção 3

No quesito desempenho, vemos que as implementações do *GRASP* dominam. Isso fica evidente tanto nas análises da Subseção 2.4 quanto da Subseção 3.4.

Já no quesito valor da solução, *Tabu Vanilla* se mostrou melhor, seguida de *GRASP Best* e *GA Vanilla*.

Assim, de forma geral, considerando tanto desempenho quanto valor da solução, chega-se a conclusão que a melhor implementação é *GRASP Best*.

Vale ressaltar que *GRASP First* apresentou melhor desempenho. Uma investigação interessante seria buscar parâmetros que melhorem o valor da solução sem prejudicar o desempenho. Se eles existirem, tal variação seria a melhor considerando todos os critérios apresentados nesse trabalho.

Referências

[1] M. G. Resende and C. C. Ribeiro, "Greedy randomized adaptive search procedures: advances and extensions," in *Handbook of metaheuristics*, pp. 169–220, Springer, 2019.

[2] M. Gendreau and J.-Y. Potvin, "Tabu search," in *Handbook of Metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), vol. 146 of *International Series in Operations Research & Management Science*, ch. 2, pp. 41–56, Springer Science+Business Media, 2010.

[3] C. R. Reeves, "Genetic algorithms," in *Handbook of metaheuristics*, pp. 109–139, Springer, 2010.

[4] R. M. Aiex, M. G. Resende, and C. C. Ribeiro, "Ttt plots: a perl program to create time-to-target plots," *Optimization Letters*, vol. 1, no. 4, pp. 355–366, 2007.

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

[6] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang, "The unconstrained binary quadratic programming problem: a survey," *Journal of combinatorial optimization*, vol. 28, no. 1, pp. 58–81, 2014.

[7] L. G. T. da Silva, "Mo824a-combinatorial-optimization," 2022. Disponível em `https://github.com/lucasguesserts/MO824A-combinatorial-optimization`.

Apêndice A *MAX-QBF com mochila* (MAX-KQBF)

Definição 1 (Conjunto Binário)**.** $\mathbb{B} = \{0, 1\}$

Definição 2 (Função Binária Quadrática (QBF))**.** É uma função $f : \mathbb{B}^n \to \mathbb{Z}$ da forma:

$$f(x) = \sum_{j=1}^{n} x_i \cdot a_{i,j} \cdot x_j = x^T \cdot A \cdot x$$

em que $a_{i,j} \in \mathbb{Z},\ \forall i, j \in \{1, \cdots, n\}$ e A é a matriz n por n induzida pelos $a_{i,j}$.

Definição 3 (Problema de Maximização de uma Função Binária Quadrática (MAX-QBF))**.** Dada uma QBF f, um MAX-QBF é um problema da forma:

$$\max_x f(x)$$

Fato 1. MAX-QBF é NP-difícil [6]

Definição 4 (Maximum knapsack quadractic binary function (MAX-KQBF))**.** Dada uma QBF f, um vetor $w \in \mathbb{Z}^n$, e um valor $W \in \mathbb{Z}$, um MAX-KQBF é um problema da forma:

$$\begin{aligned} \max \quad & f(x) \\ \text{subjected to} \quad & w^T x \leq W \\ & x \in \mathbb{B}^n \end{aligned}$$

A.1 Instâncias

Foram utilizadas as instâncias com características conforme a Tabela 4.

Instância	Num. Variáveis	Num. Possibilidades	Intervalo de Otimalidade
kqbf020	20	1.0e+06	[80, 151]
kqbf040	40	1.1e+12	[275, 429]
kqbf060	60	1.2e+18	[446, 576]
kqbf080	80	1.2e+24	[729, 1000]
kqbf100	100	1.3e+30	[851, 1539]
kqbf200	200	1.6e+60	[3597, 5826]
kqbf400	400	2.6e+120	[10846, 16625]

Tabela 4: Instâncias do problema MAX-KQBF. Os dados completos estão disponíveis em [7].

A.2 *GRASP*

Algorithm 1 *GRASP*

1: $S_{\text{best}} \leftarrow \varnothing$
2: **for** $k = 1, \ldots, N_{it}$ **do**
3: $\quad S \leftarrow$ Greedy-Randomized-Construction()
4: $\quad S \leftarrow$ Local-Search(S)
5: $\quad S_{\text{best}} \leftarrow \max\{S, S_{\text{best}}\}$
6: **return** S_{best}

Algorithm 2 Greedy-Randomized-Construction(α)

$S \leftarrow \varnothing$
$C \leftarrow E$ ▷ Candidates list
for $e \in C$ **do**
    $c(e) \leftarrow$ Incremental-Cost(e, S) ▷ Increment in the cost by adding e to S
while $C \neq \varnothing$ **do**
    $c_{min} = \min\{c(e) : e \in C\}$
    $c_{max} = \max\{c(e) : e \in C\}$
    $R \leftarrow \{e \in C : c(e) \leqslant c_{min} + \alpha(c_{max} - c_{min})\}$ ▷ Restricted candidates list
    $s \leftarrow$ Select-Random-Element(R)
    $S \leftarrow S \cup \{s\}$
    Update C
    Update $c(e)$
return S

Algorithm 3 Loca-Search(S)

while S is not local optimal **do**
    $S \leftarrow \arg\max_{S' \in N(S)} \{f(S')\}$ ▷ $N(S)$ is the neighborhood of S ▷ f is the goal function
return S

A.3 *Tabu*

Algorithm 4 *Tabu* (S_0)

$S \leftarrow S_0$ ▷ S_0 is the initial solution ▷ S is the current solution
$S^* \leftarrow S$ ▷ S^* is the current best solution
$f^* \leftarrow f(S^*)$ ▷ f is the goal function
$T \leftarrow \varnothing$ ▷ T is the tabu list
while not Termination-Criteria-Satisfied **do**
    $S \leftarrow \arg\max_{S' \in N(S,T)} \{f(S')\}$ ▷ $N(S,T)$ is the neighborhood of S limited by T
    if $f(S) > f^*$ **then**
        $S^* \leftarrow S$
        $f^* \leftarrow f(S)$
    Update T ▷ record moves and delete old entries
return S^*

A.4 *GA*

Algorithm 5 *GA*

```
escolher a população inicial de cromossomos
while condição de parada não é satisfeita do
    while Não há descendentes suficientes do
        if condição de cruzamento é satisfeita then
            Selecionar cromossomos mãe
            Selecionar parâmetros de cruzamento
            Executar cruzamento
        if condição de mutação é satisfeita then
            Selecionar pontos de mutação
            Executar mutação
        Calcular adaptação (fitness) dos descendentes
    Selecionar nova população
return Melhor cromossomo da população
```

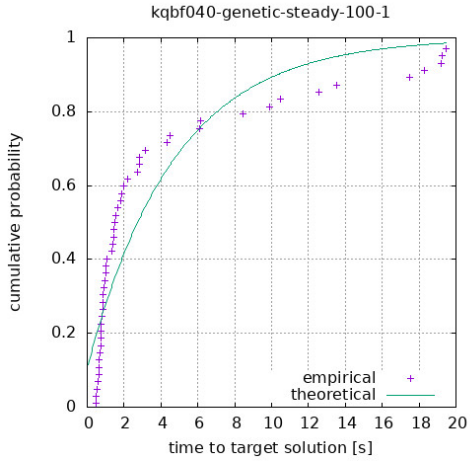
Apêndice B Implementação e execução dos experimentos

O programs foram executados num ideapad S145 81S90005BR: Lenovo IdeaPad S145 Notebook Intel Core i5-8265U (6MB Cache, 1.6GHz, 8 cores), 8GB DDR4-SDRAM, 460 GB SSD, Intel UHD Graphics 620 no ambiente:

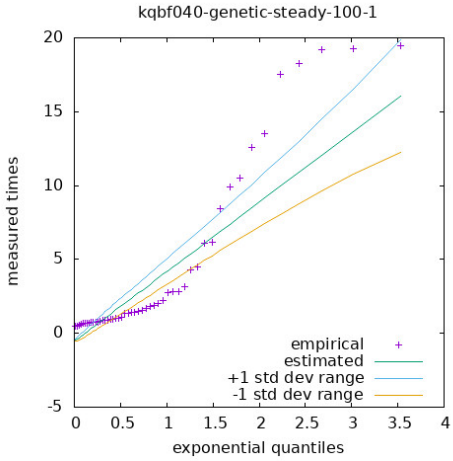
1. sistema operacional: Fedora 35
2. Java versão 17
3. Gradle versão 7.4

O desenvolvimento da solução do problema foi feito em Java, baseado nos frameworks disponibilizados pelos professores. O código pode ser encontrado em [7].

Apêndice C Time-To-Target Plot (TTT Plot)

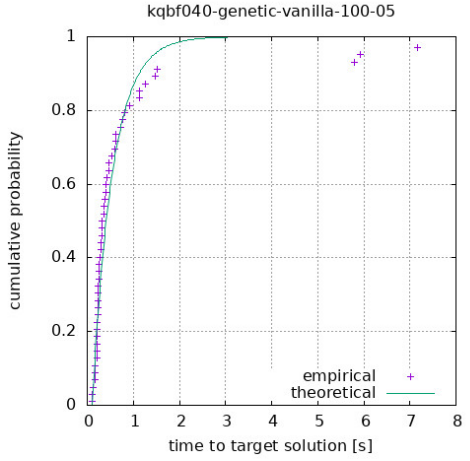


(a) Cumulative Probability Distribution - Algorithm GA steady - Problem kqbf040

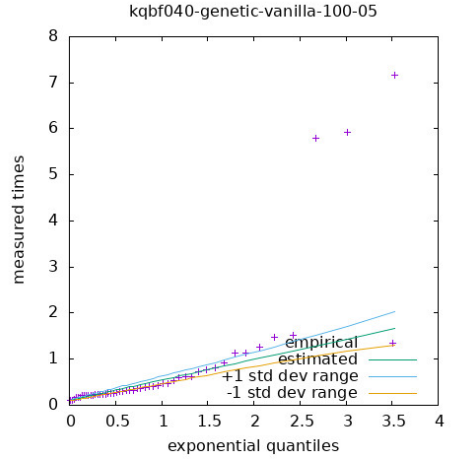


(b) Q-Q plot - Algorithm GA steady - Problem kqbf040

Figura 1: Algorithm GA steady - Problem kqbf040.

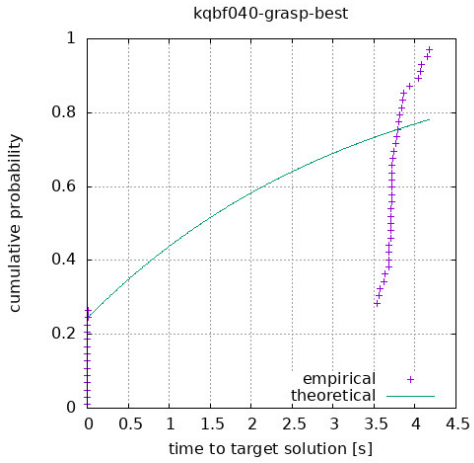


(a) Cumulative Probability Distribution - Algorithm GA vanilla - Problem kqbf040

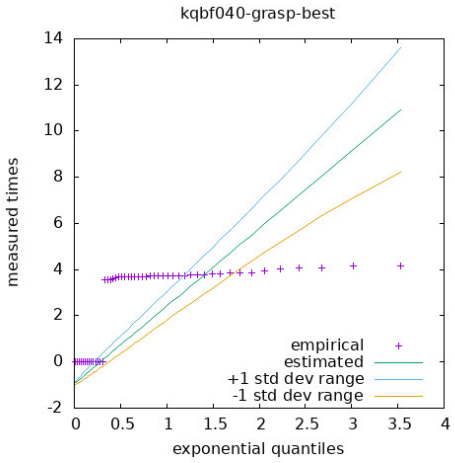


(b) Q-Q plot - Algorithm GA vanilla - Problem kqbf040

Figura 2: Algorithm GA vanilla - Problem kqbf040.

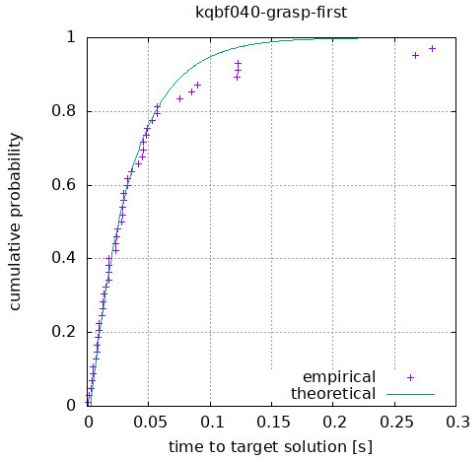


(a) Cumulative Probability Distribution - Algorithm GRASP Best - Problem kqbf040

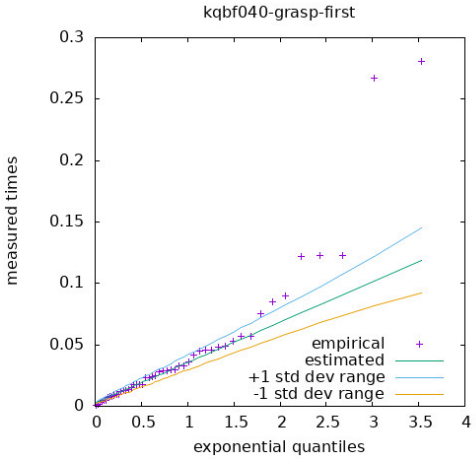


(b) Q-Q plot - Algorithm GRASP Best - Problem kqbf040

Figura 3: Algorithm GRASP Best - Problem kqbf040.

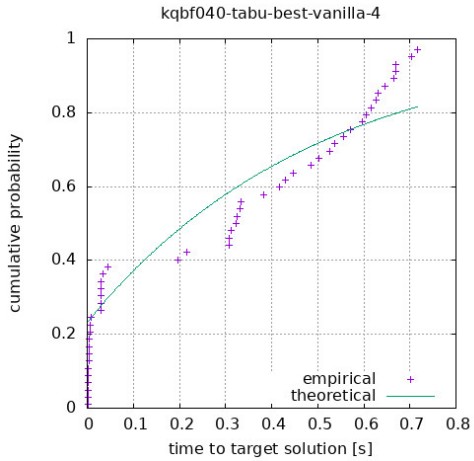


(a) Cumulative Probability Distribution - Algorithm GRASP First - Problem kqbf040

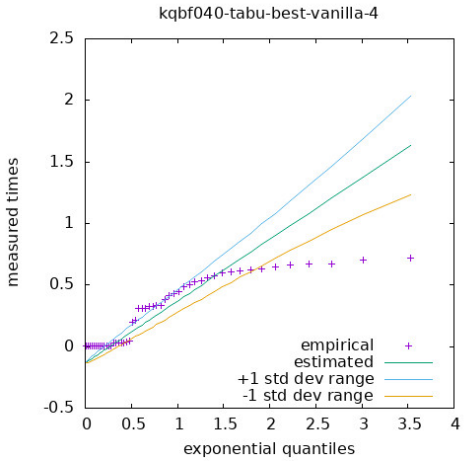


(b) Q-Q plot - Algorithm GRASP First - Problem kqbf040

Figura 4: Algorithm GRASP First - Problem kqbf040.

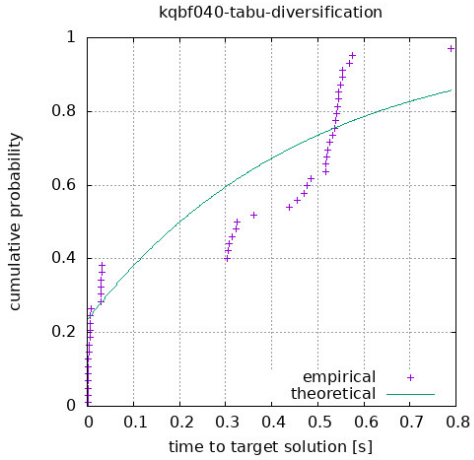


(a) Cumulative Probability Distribution - Algorithm Tabu vanilla - Problem kqbf040

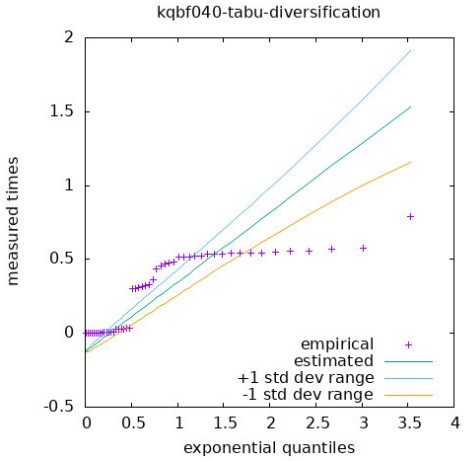


(b) Q-Q plot - Algorithm Tabu vanilla - Problem kqbf040

Figura 5: Algorithm Tabu vanilla - Problem kqbf040.

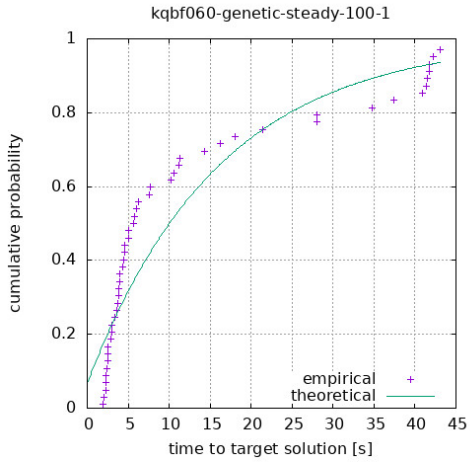


(a) Cumulative Probability Distribution - Algorithm Tabu com Intensificação e Diversificação - Problem kqbf040

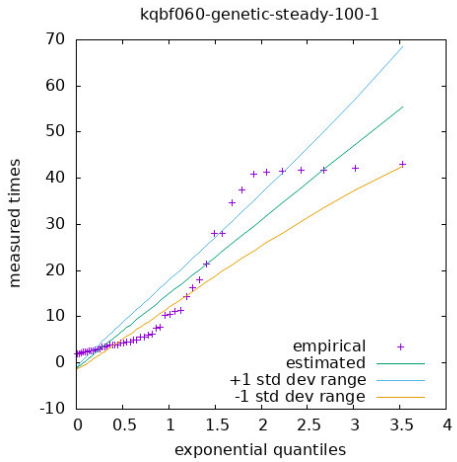


(b) Q-Q plot - Algorithm Tabu com Intensificação e Diversificação - Problem kqbf040

Figura 6: Algorithm Tabu com Intensificação e Diversificação - Problem kqbf040.

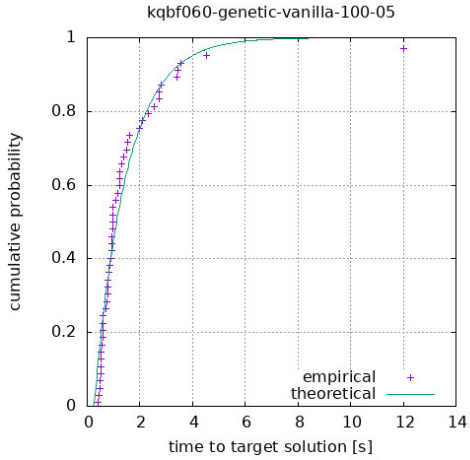


(a) Cumulative Probability Distribution - Algorithm GA steady - Problem kqbf060

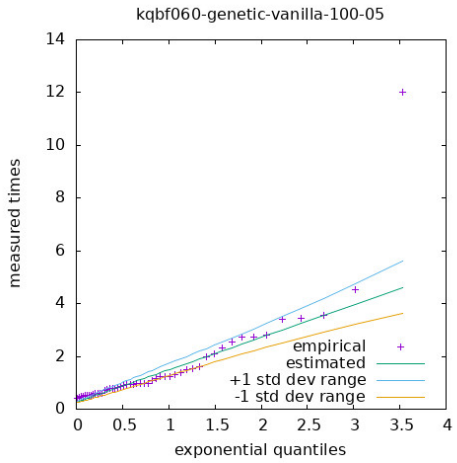


(b) Q-Q plot - Algorithm GA steady - Problem kqbf060

Figura 7: Algorithm GA steady - Problem kqbf060.

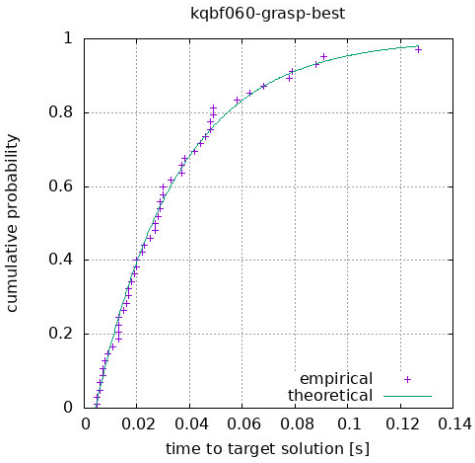


(a) Cumulative Probability Distribution - Algorithm GA vanilla - Problem kqbf060

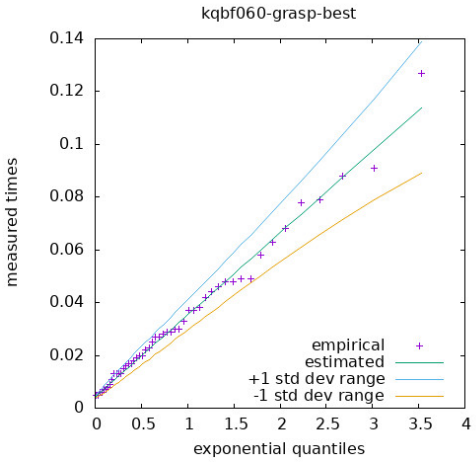


(b) Q-Q plot - Algorithm GA vanilla - Problem kqbf060

Figura 8: Algorithm GA vanilla - Problem kqbf060.

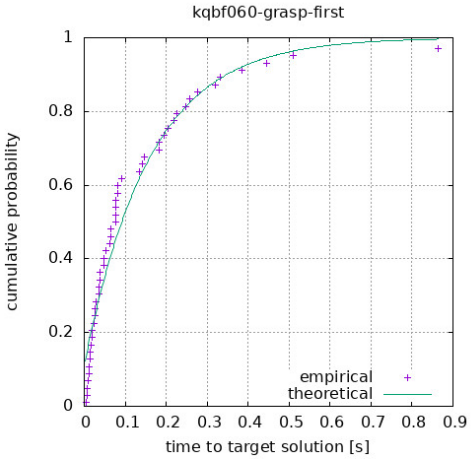


(a) Cumulative Probability Distribution - Algorithm GRASP Best - Problem kqbf060

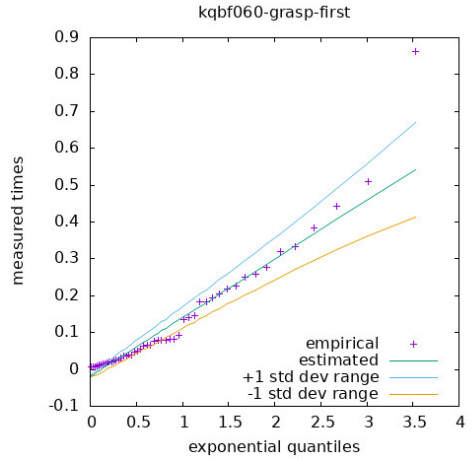


(b) Q-Q plot - Algorithm GRASP Best - Problem kqbf060

Figura 9: Algorithm GRASP Best - Problem kqbf060.

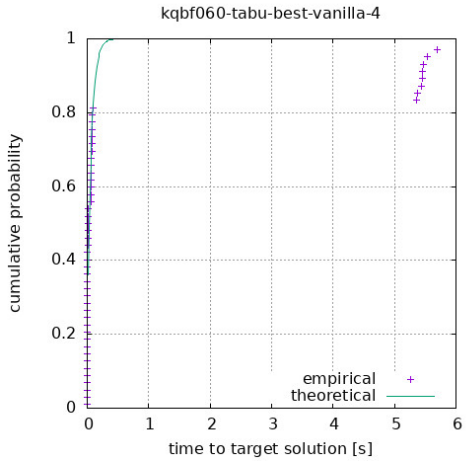


(a) Cumulative Probability Distribution - Algorithm GRASP First - Problem kqbf060

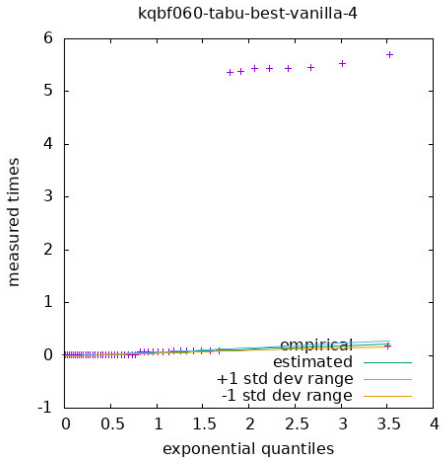


(b) Q-Q plot - Algorithm GRASP First - Problem kqbf060

Figura 10: Algorithm GRASP First - Problem kqbf060.

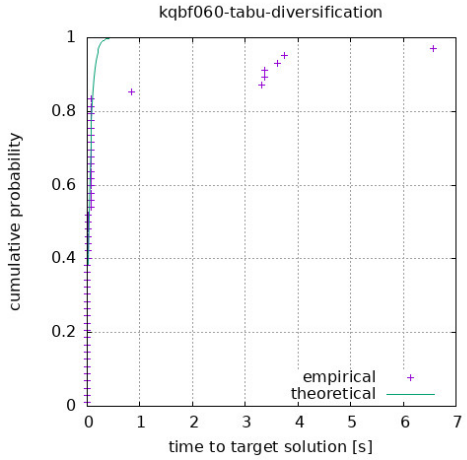


(a) Cumulative Probability Distribution - Algorithm Tabu vanilla - Problem kqbf060

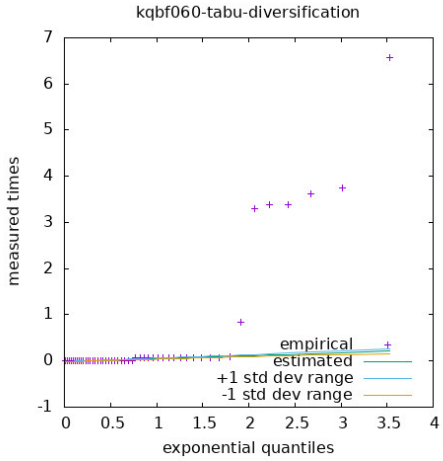


(b) Q-Q plot - Algorithm Tabu vanilla - Problem kqbf060

Figura 11: Algorithm Tabu vanilla - Problem kqbf060.

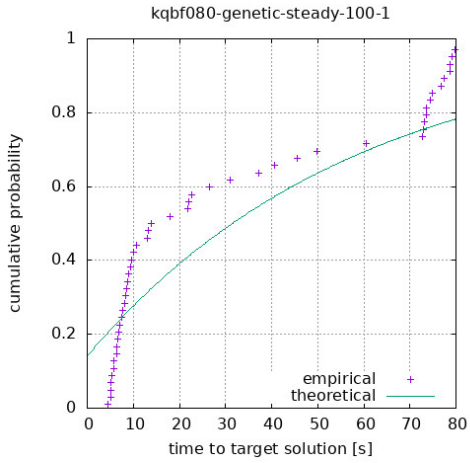


(a) Cumulative Probability Distribution - Algorithm Tabu com Intensificação e Diversificação - Problem kqbf060

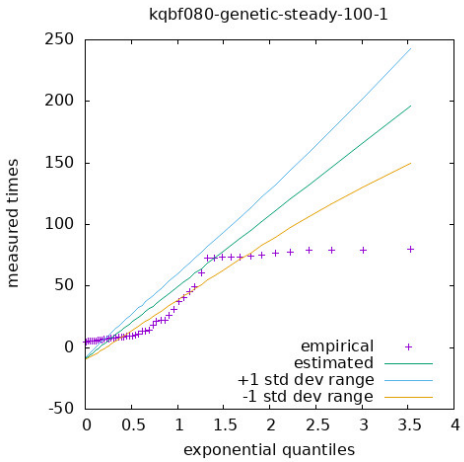


(b) Q-Q plot - Algorithm Tabu com Intensificação e Diversificação - Problem kqbf060

Figura 12: Algorithm Tabu com Intensificação e Diversificação - Problem kqbf060.

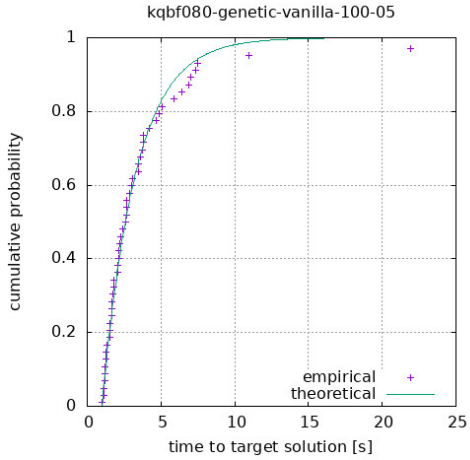


(a) Cumulative Probability Distribution - Algorithm GA steady - Problem kqbf080

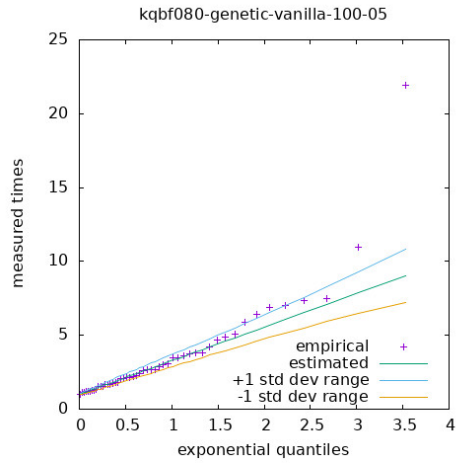


(b) Q-Q plot - Algorithm GA steady - Problem kqbf080

Figura 13: Algorithm GA steady - Problem kqbf080.

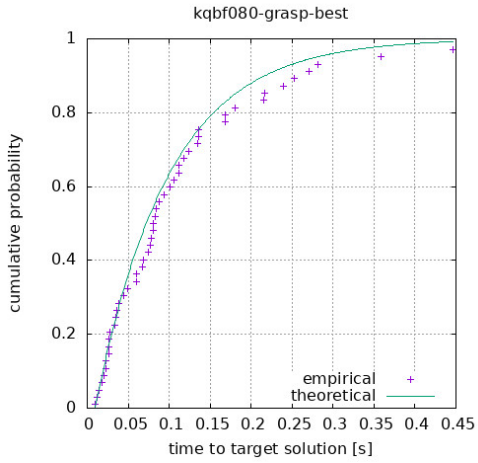


(a) Cumulative Probability Distribution - Algorithm GA vanilla - Problem kqbf080

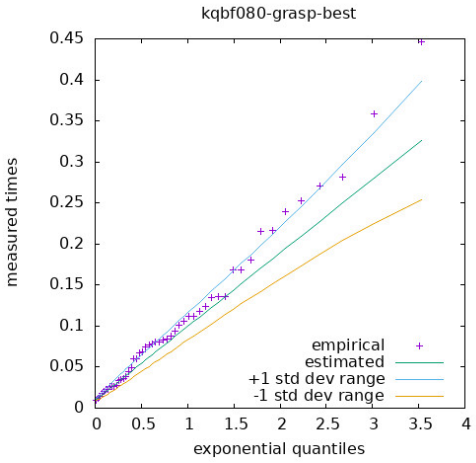


(b) Q-Q plot - Algorithm GA vanilla - Problem kqbf080

Figura 14: Algorithm GA vanilla - Problem kqbf080.

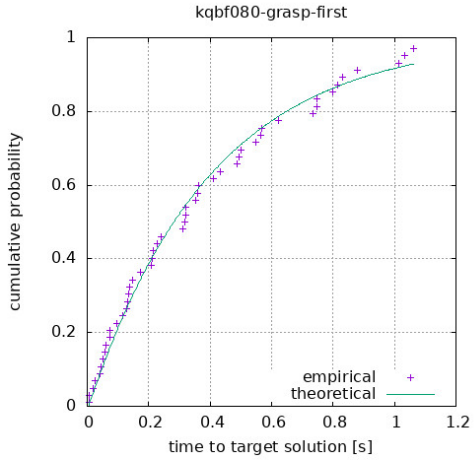


(a) Cumulative Probability Distribution - Algorithm GRASP Best - Problem kqbf080

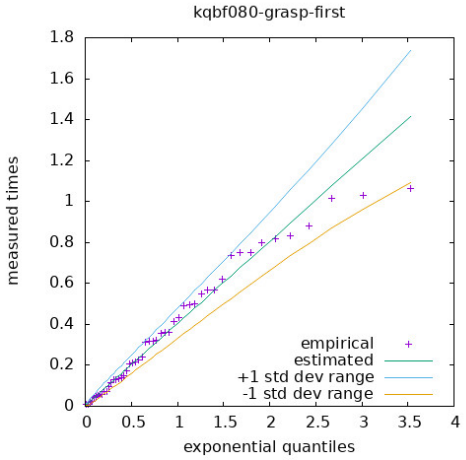


(b) Q-Q plot - Algorithm GRASP Best - Problem kqbf080

Figura 15: Algorithm GRASP Best - Problem kqbf080.

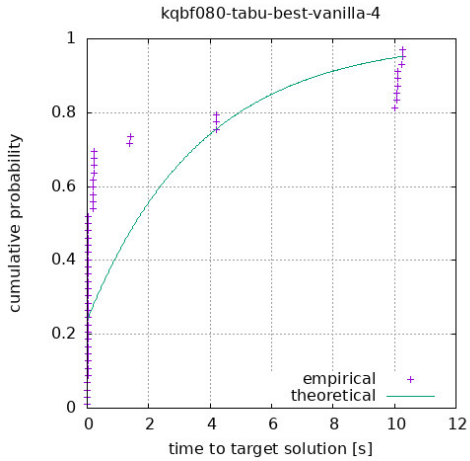


(a) Cumulative Probability Distribution - Algorithm GRASP First - Problem kqbf080

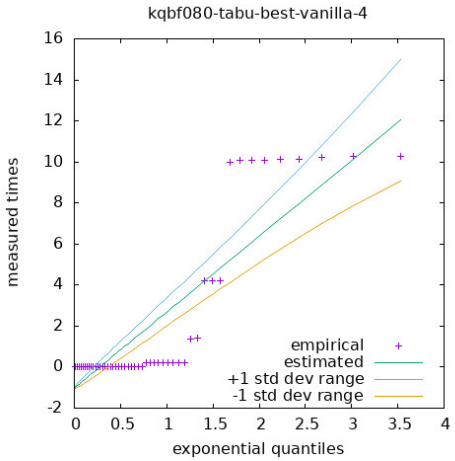


(b) Q-Q plot - Algorithm GRASP First - Problem kqbf080

Figura 16: Algorithm GRASP First - Problem kqbf080.

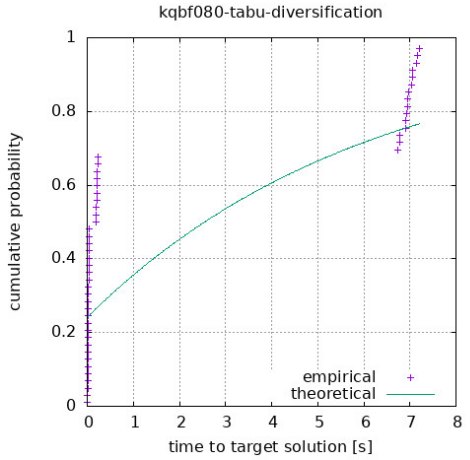


(a) Cumulative Probability Distribution - Algorithm Tabu vanilla - Problem kqbf080

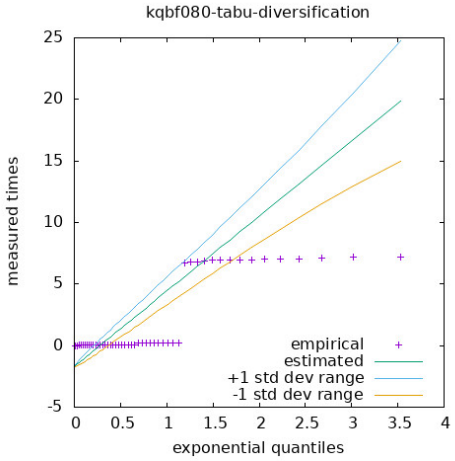


(b) Q-Q plot - Algorithm Tabu vanilla - Problem kqbf080

Figura 17: Algorithm Tabu vanilla - Problem kqbf080.



(a) Cumulative Probability Distribution - Algorithm Tabu com Intensificação e Diversificação - Problem kqbf080



(b) Q-Q plot - Algorithm Tabu com Intensificação e Diversificação - Problem kqbf080

Figura 18: Algorithm Tabu com Intensificação e Diversificação - Problem kqbf080.

Apêndice D *Performance Profile*

Problema	kqbf020	kqbf040	kqbf060	kqbf080	kqbf100	kqbf200	kqbf400
GRASP First	0.145	0.353	1.261	3.016	6.056	75.552	inf
GRASP Best	0.074	1.765	1.981	4.242	9.068	100.731	inf
Tabu Vanilla	0.997	2.707	5.427	10.318	17.549	160.816	inf
Tabu com Intensificação e Diversificação	0.255	1.218	5.662	inf	17.294	444.594	inf
GA Vanilla	0.974	3.200	5.529	9.627	15.386	62.351	inf
GA Steady	3.015	10.860	23.954	46.665	76.459	254.404	inf

Tabela 5: Dados utilizados na confecção do *Performance Profile*. *inf* acima significa que o algoritmo não consegui resolver o problema, isto é, não encontrou uma solução com valor pelo menos o valor definido para o problema em questão.

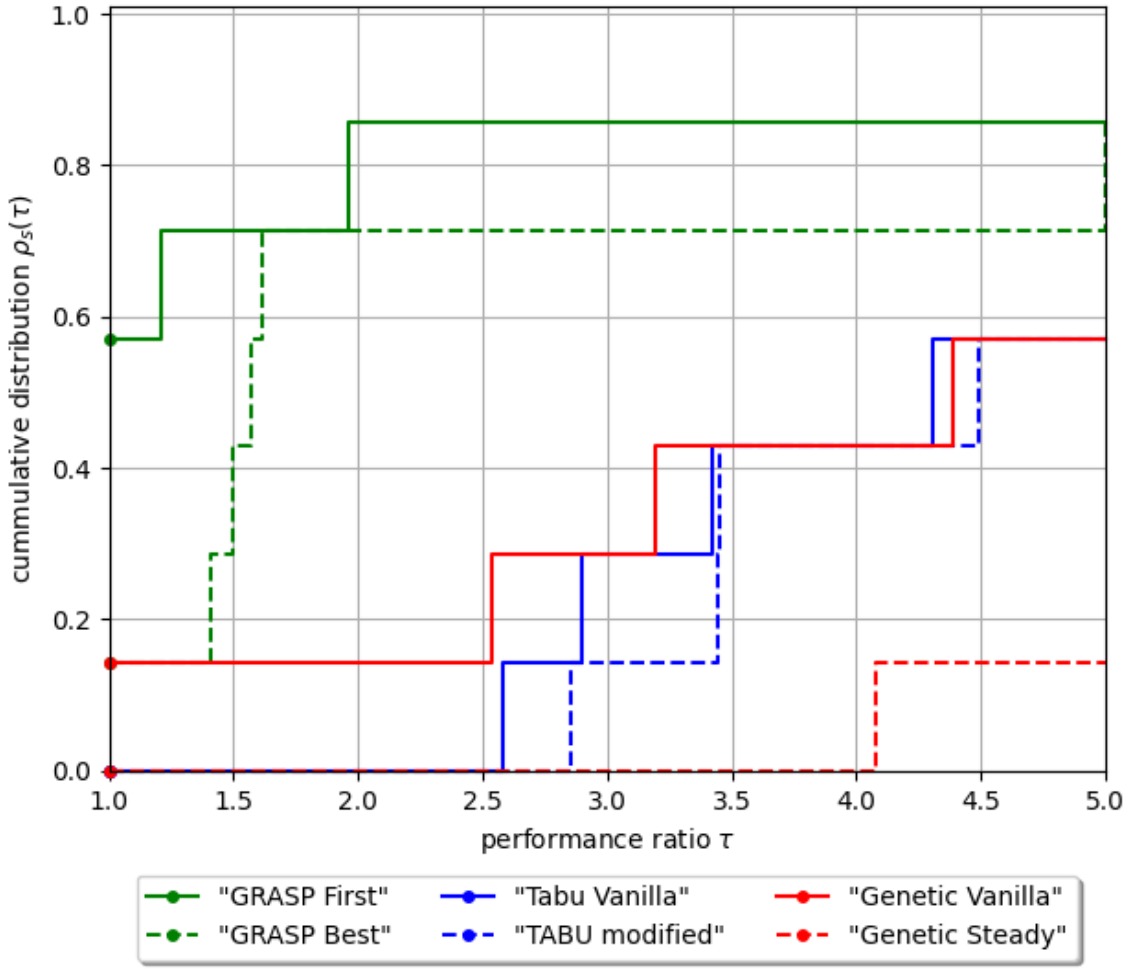


Figura 19: Gráfico de *Performance Profile*no intervalo $[1, 5]$

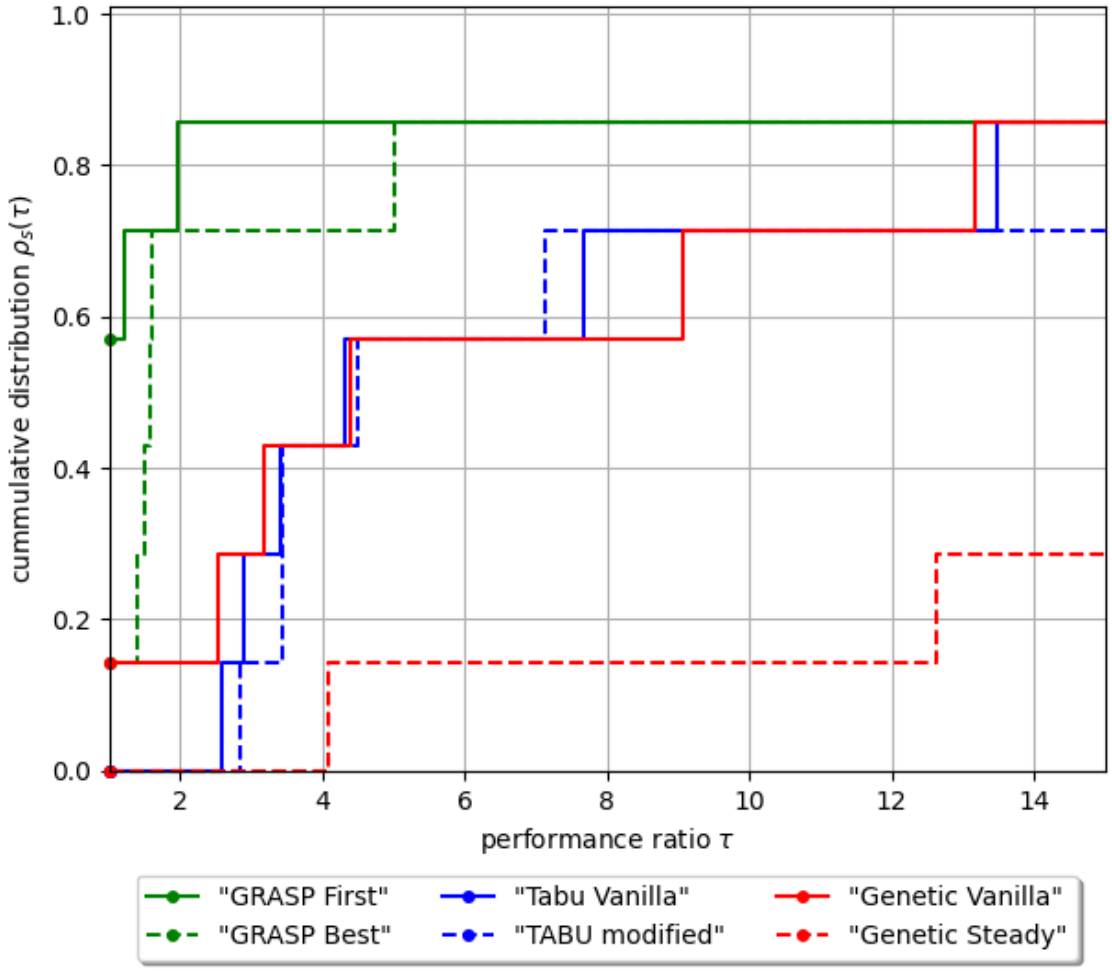


Figura 20: Gráfico de *Performance Profile*no intervalo $[1, 15]$

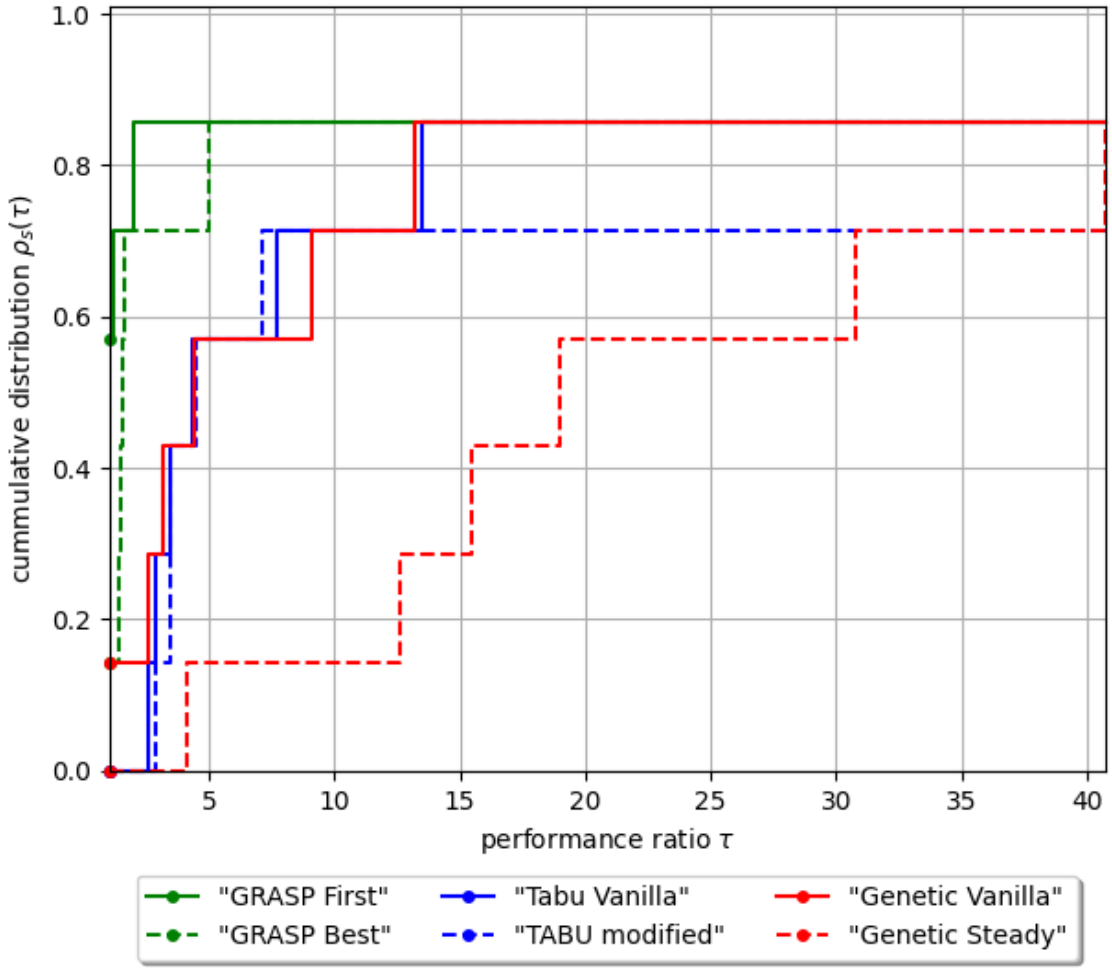


Figura 21: Gráfico de *Performance Profile*no intervalo r_M (calculado automaticamente pelo software de plot).