Relatório referente ao primeiro trabalho de implementação da matéria de Inteligência Artificial

Rafael Belmock Pedruzzi

Universidade Federal do Espírito Santo - Vitória

Abstract

Neste relatório realizaremos uma comparação de desempenho entre diversas meta-heurísticas aplicadas ao problema da mochila.

1. Introdução

Os vários avanços no ramo da inteligência artificial que vem ocorrendo nas ultimas décadas geraram um grande número de técnicas genéricas de resolução de problemas (meta-heurísticas) que podem ser aplicadas a uma grande variedade de situações. Porém essas diferentes meta-heurísticas possuem diferentes formas de resolução e implementação, o que leva a variações na sua eficiência quando aplicadas a determinados problemas. Neste relatório iremos realizar uma comparação de desempenho entre as seguintes meta-heurísticas: Escalada (Hill Climbing), Pesquisa por Feixe (Beam Seach), Busca Adaptativa Randômica Gulosa (Greedy Randomized Adaptive Search - GRASP), Resfriamento Simulado (Simulated Annealing) e o Algorítimo Genético (Genetic Algorithm) aplicadas a algumas instâncias do problema da mochila com repetição de objetos.

Este relatório esta organizado da seguinte forma: a sessão 2 apresenta o problema da mochila e as características particulares especificas do problema utilizado. A sessão 3 descreve as meta-heurísticas comparadas e traços gerais de sua implementação. A sessão 4 detalha os experimentos realizados, mostrando os algorítimos, as instâncias do problema utilizadas e os resultados obtidos. Finalmente, a sessão 5 apresenta a analise geral dos resultados e suas conclusões.

2. O Problema da Mochila

Definido em 1972 por Richard Karp como um de seus 21 problemas NP-completos, o problema da mochila (ou knapsack problem, em inglês) é um problema de otimização onde é necessário preencher uma mochila com objetos de diferentes pesos e valores. O objetivo é que se preencha a mochila com o maior valor possível, não ultrapassando o peso máximo. Para este teste será utilizada uma versão do problema com repetição de objetos, ou seja, cada objeto poderá ser colocado na mochila um número indefinido de vezes.

O problema é formulado da seguinte forma: Seja T o tamanho máximo da mochila e VT um conjunto de n objetos $x_1, ..., x_n$ com pesos $t_1, ..., t_n$ e valores $v_1, ..., v_n$, respectivamente. Deseja-se maximizar a equação:

$$\sum_{1 \leq i \leq n} v_i x_i, \; sujeito \; a \; \sum_{1 \leq i \leq n} t_i x_i \leq T onde \; x_i \in \mathbb{N}$$

3. Os Métodos Utilizados

A seguir serão explicadas as meta-heurísticas utilizadas e como foram adaptadas ao problema da mochila. Antes, porem, explicitaremos alguns elementos comuns a todas elas referente a implementação do problema. São eles estados, valor, tamanho, validade, expansão e regressão de um estado.

Um estado é uma possível solução do problema em questão que será gradualmente refinada. Para o problema da mochila, um estado foi implementado como uma lista com n números inteiros, representando a quantidade de cada elemento x_i atualmente na mochila. Vale ressaltar que o estado inicial para todas as meta-heurísticas foi a mochila vazia, ou seja, $x_i = 0$ para todo $0 \leq i \leq n$. O valor e tamanho de um estado são, respectivamente, a soma dos valores e a soma dos tamanhos de cada objeto no estado. A validade de um estado é um valor booleano que indica se um estado cumpre os requerimentos necessários para ser uma solução. No caso deste problema, um estado é dito válido se o tamanho do estado não ultrapassa o limite da mochila (T). A expansão refere-se a todos os possíveis acréscimos unitários do estado expandido, ou seja, dado um estado e sua expansão será $e + x$ para cada x em $x_1, ..., x_n$. Finalmente, a regressão é análoga a expansão porem referente aos possíveis decréscimos unitários do estado (e - x para cada x em $x_1, ..., x_n$).

3.1. Escalada

A mata-heurística Escalada é uma abordagem gulosa baseada em um estado inicial e uma função estimativa heurística. A cada iteração, analisa-se as possíveis expansões do estado atual por meio da função estimativa. A expansão mais bem avaliada será o próximo estado. Esse processo se repete até que não hajam mais expansões válidas.

Como citado anteriormente o estado inicial será a mochila vazia. Já para a função estimativa, será selecionado da expansão o estado válido com maior valor. Quando não houverem mais estados válidos na expansão a heurística encerrará retornando o estado atual.

3.2. Pesquisa por Feixe

Esse método é similar à meta-heurística Escalada porem mantendo não somente o melhor estado mas sim os m melhores, dividindo o processo de busca em m ramos distintos. No fim, os estados encontrados por cada ramo são comparados e o melhor entre eles é selecionado. Isso permite um melhor aproveitamento do espaço de busca, aumentando a chance de evitar máximos (já que esse é um problema de maximização) locais.

Para a implementação foi utilizada uma estrutura similar a da Escalada com duas modificações significativas: A função estimativa selecionará não só o melhor estado mas sim os m melhores (ou todos os possíveis, caso hajam menos do que m) e foi implementada uma fila de estados inicialmente contendo apenas o estado inicial. A cada iteração um estado é retirado da fila, expandido e as melhores expansões são adicionadas ao fim da fila. Devido a função estimativa, somente estados válidos da expansão são selecionados. Dessa forma, quando a fila estiver vazia a heurística encerrará e retornará o melhor estado encontrado.

3.3. GRASP

A meta-heurística GRASP pode ser dividida em duas fases. Na primeira será gerada uma solução pseudo-aleatória a partir de um algorítimo guloso e na segunda será realizada uma busca local a partir dessa solução para tentar otimizar o resultado. O método de busca local utilizado foi o método de Descida Profunda (Deepest Descent), que consiste em migrar para o melhor estado "vizinho"do estado atual encontrado.

A primeira fase foi implementada de forma análoga a meta-heurística de Escalada porém sua função estimativa, ao invés de selecionar o melhor estado da expansão, seleciona aleatoriamente um dentre os m melhores. A seleção aleatória foi feita com o método da roleta, que faz os melhores estados terem chances maiores de serem selecionados.

Na segunda fase simplesmente será realizada uma Descida Profunda da solução encontrada na primeira fase para maximiza-la. Para a seleção dos vizinhos de um estado foi inicialmente cogitado compor-los da expansão do estado

atual e da redução de todos os estados da expansão e do estado atual. Porem essa implementação mostrou-se inviável devido ao alto consumo de memória. Por isso optou-se por utilizar somente os estados da expansão e da redução do estado atual.

3.4. Resfriamento Simulado

O Resfriamento Simulado é uma meta-heurística baseada no processo físico do resfriamento de um sólido superaquecido. Conforme esfria, o sólido gradativamente alcança estados de menor energia. As vezes, de forma aleatória, o sólido retorna momentaneamente para estados de maior energia, um evento que torna-se menos frequente conforme esfria.

Computacionalmente, o resfriamento pode ser comparado com o processo de aproximação da solução de um problema. A cada iteração um estado aleatório da vizinhança do estado atual será avaliado. Caso esse estado seja melhor que o atual ele será selecionado e a execução avançará a partir dele. Caso contrário haverá uma chance aleatória dele ser selecionado baseada em um fator "temperatura". Conforme as iterações avançam a temperatura será gradualmente reduzida em um fator α até que se atinja o critério de parada, que, para esta implementação, será o número máximo de iterações. Para a função de vizinhança foi utilizada a mesma do algorítimo de Descida Profunda.

3.5. Algorítimo Genético

Baseado na teoria de seleção natural e hereditariedade, esta meta-heurística simula a evolução natural das espécies onde os indivíduos mais aptos persistirão enquanto os demais serão descartados. O algorítimo parte de uma população inicial composta de m estados aleatoriamente gerados. A cada iteração os melhores estados serão selecionados e utilizados para gerar uma nova população. O processo de gerar novas populações utiliza duas operações principais. São elas o crossover, onde dois estados serão combinados (literalmente trocando partes entre sí) para gerar dois novos estados, e a mutação, em que posições aleatórias

de um estado são levemente alteradas. A condição de parada desse algorítimo pode variar mas neste caso será utilizado o número máximo de gerações

O operador de crossover foi implementado de forma que uma partição de tamanho aleatório entre 1 e $n-1$ posições partindo do primeiro elemento de um estado fosse trocada com o mesmo trecho do segundo estado. Em outras palavras, dado que $e_1, ..., e_n$ são as quantidades de cada objeto $x_1, ..., x_n$ em um estado, um trecho $e_1, ..., e_{k<n}$ será trocado entre dois estados $E_1 = e1_1, ..., e1_n$ e $E_2 = e2_1, ..., e2_n$ gerando os estados $E3 = e1_1, ..., e1_k, e2_{k+1}, ..., e2_n$ e $E4 = e2_1, ..., e2_k, e1_{k+1}, ..., e1_n$.

Quanto ao operador de mutação, inicialmente será selecionado aleatoriamente um número de alterações k a serem realizadas entre 1 e n. Em seguida serão feitas k alterações no estado, selecionando um objeto x_i aleatório dele e incrementando o número de ocorrência desse objeto em 1 ou decrementando-o em 1 (até o mínimo de 0) com uma probabilidade de 50%.

A população inicial foi gerada com um algorítimo simples. Inicialmente o estado inicial (mochila vazia) é inserido na população e então o algorítimo entra em um loop que seleciona o último estado inserido e realiza sobre ele o operador de mutação. Isso é repetido até que seja alcançado o número desejado de indivíduos.

Por fim, para gerar cada indivíduo de uma nova população aplicam-se os seguintes passos: Primeiramente um indivíduo da população atual é selecionado pelo supracitado método da roleta. Esse individuo, então, possui uma certa probabilidade de sofrer o operador de crossover (nesse caso outro indivíduo será retirado da população atual para servir de par) e/ou o operador de mutação. Independentemente do(s) operador(es) executado(s), o(s) estado(s) resultante(s) da(s) operação(es) serão adicionados a nova população, ou o mesmo estado selecionado caso não nenhuma seja aplicada. Além disso, para evitar que a melhor solução encontrada se perca, o melhor estado da população atual será automaticamente preservado na nova população. Isso é uma operação opcional do algorítimo conhecida como elitismo, no caso aplicado a apenas um indivíduo.

4. Os Experimentos Realizados

Nessa seção descreveremos os experimentos e a forma como foram realizados. Na primeira seção apresentaremos o algorítimo e problemas da etapa de treino das meta-heurísticas e também os resultados obtidos com eles. Na segunda seção apresentaremos o algorítimo, problemas e resultados dos testes.

4.1. Etapa de Treino

Podemos perceber que, com exceção da Escalada, todas as demais meta-heurísticas possuem um ou mais hiperparâmetros que, se escolhidos de forma incorreta, podem alterar seu desempenho. Para que nenhuma meta-heurística seja prejudicada na comparação devemos ajusta-las com os hiperparâmetros mais apropriados. Para tal foram selecionados 10 problemas de treino que serão exaustivamente executados em todas elas com diversos hiperparâmetros distintos a fim de selecionar os mais apropriados para cada meta-heurística.

Além disso, para evitar gasto desnecessário de tempo, o tempo de execução de cada meta-heurística foi limitado a 2 minutos.

4.1.1. Algoritmo de Treino

Foi utilizado o seguinte código para treinamento das meta-heurísticas:

4.1.2. Problemas de Treino

A seguir apresentaremos as 10 instâncias do problema da mochila selecionadas para a execução do algorítimo apresentado na sessão anterior.

1. P1:
   Tamanho da Mochila: 19
   Lista de Itens ((vi,ti)):
   (1,3),(4,6),(5,7)
2. P3:
   Tamanho da Mochila: 58
   Lista de Itens ((vi,ti)):
   (1,3),(4,6),(5,7),(3,4)

Algorithm 1 Algorítimo de Treino

```
1: função TREINO
2:     para cada meta-heurística que precisa de ajuste faça
3:         para cada problema no conjunto de treino faça
4:             para cada combinação dos valores de hiperparâmetros faça
5:                 executar meta-heurística no problema
6:                 armazenar valor da solução e tempo de execução
7:             fim para
8:             normalizar os valores encontrados com essa configuração
9:         fim para
10:        para cada combinação dos valores de hiperparâmetros faça
11:            normalizar valores encontrados
12:            se media dos resultados normalizados é a maior então
13:                armazenar valores de hiperparâmetros para uso no teste
14:            fim se
15:        fim para
16:        gerar boxplot dos resultados alcançados pela meta-heurística
17:        gerar boxplot dos tempos alcançados pela meta-heurística
18:    fim para
19: fim função
```

3. P4:

   Tamanho da Mochila: 58

   Lista de Itens ((vi,ti)):

   (1,3),(4,6),(5,7),(3,4),(8,10),(4,8),(3,5),(6,9)

4. P6:

   Tamanho da Mochila: 58

   Lista de Itens ((vi,ti)):

   (1,3),(4,6),(5,7),(3,4),(8,10),(4,8),(3,5),(6,9),(2,1)

5. P8:

   Tamanho da Mochila: 120

   Lista de Itens ((vi,ti)):

   (1,2),(2,3),(4,5),(5,10),(14,15),(15,20),(24,25),(29,30),(50,50)

6. P9:

   Tamanho da Mochila: 120

   Lista de Itens ((vi,ti)):

   (1,2),(2,3),(3,5),(7,10),(10,15),(13,20),(24,25),(29,30),(50,50)

7. P11:

   Tamanho da Mochila: 120

   Lista de Itens ((vi,ti)):

   (24,25),(29,30),(50,50)

8. P14:

   Tamanho da Mochila: 138

   Lista de Itens ((vi,ti)):

   (1,3),(4,6),(5,7),(3,4),(2,6),(2,3),(6,8),(1,2),(2,3),(3,5),(7,10), (10,15),(13,20),(24,25),(29,30),(50,50)

9. P17:

   Tamanho da Mochila: 13890000

   Lista de Itens ((vi,ti)):

   (1,3),(4,6),(5,7),(3,4),(2,6),(2,3),(6,8),(1,2),(3,5),(7,10),(10,15),(13,20),(24,25),(29,37)

10. P20:

    Tamanho da Mochila: 45678901

Lista de Itens ((vi,ti)):
(1,3),(4,6),(5,7),(3,4),(2,6),(1,2),(3,5),(7,10),(10,15),(13,20),(15,20)

4.1.3. Apresentação e Analise dos Hiperparâmetros

Aqui apresentaremos os hiperparâmetros utilizados e os resultados obtidos pelo algorítimo de treino. Serão mostrados os valores de hiperparâmetros escolhidos e os boxplots dos valores obtidos e tempo de execução das 10 melhores combinações de hiperparâmetros para cada meta-heurística (com exeção da Pesquisa por Feixe que possui apenas 4 possíveis combinações).

Os valores de hiperparâmetros testados foram:

- Pesquisa por Feixe:
  quantidade de ramos = $[10, 25, 50, 100]$
- GRASP:
  numero máximo de iterações = $[50, 100, 200, 350, 500]$
  numero de melhores elementos = $[2, 5, 10, 15]$
- Resfriamento Simulado:
  temperatura inicial = $[500, 100, 50]$
  fator $\alpha = [0.95, 0.85, 0.7]$
  numero máximo de iterações = $[350, 500]$
- Algorítimo Genético:
  tamanho da população = $[10, 20, 30]$
  taxa de crossover = $[0.75, 0.85, 0.95]$
  taxa de mutação = $[0.10, 0.20, 0.30]$

Desses, os melhores encontrados para cada meta-heurística foram:

- Pesquisa por Feixe:
  quantidade de ramos = 10
- GRASP:
  numero máximo de iterações = 500
  numero de melhores elementos = 15

- Resfriamento Simulado:

  temperatura inicial = 500

  fator $\alpha = 0.95$

  numero máximo de iterações = 500

- Algorítimo Genético:

  tamanho da população = 30

  taxa de crossover = 0.95

  taxa de mutação = 0.30

Sendo que, para suas 10 melhores combinações com relação a média dos resultados obtidos, os resultados e tempos de execução alcançados foram, em ordem decrescente:

Figura 1: Pesquisa por Feixe - Valores Normalizados

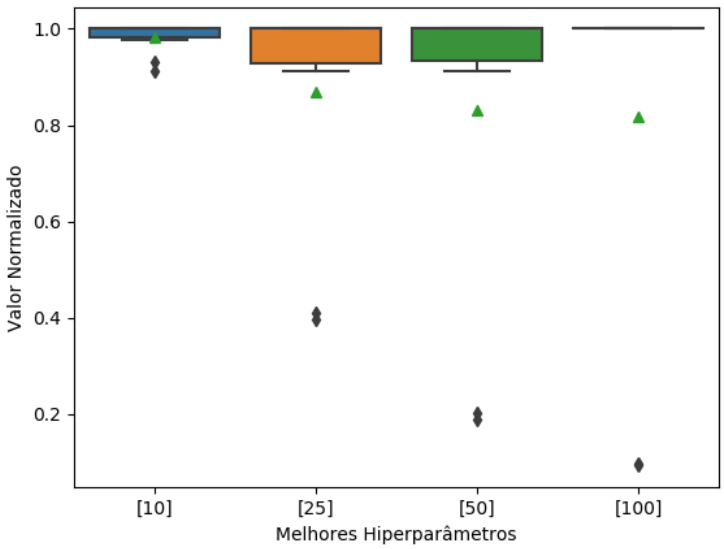


Figura 2: Pesquisa por Feixe - Tempo de Execução

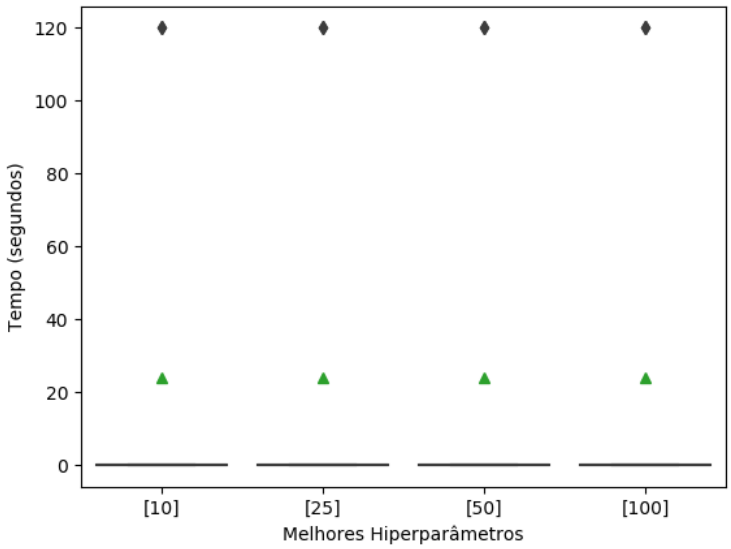


Figura 3: GRASP - Valores Normalizados

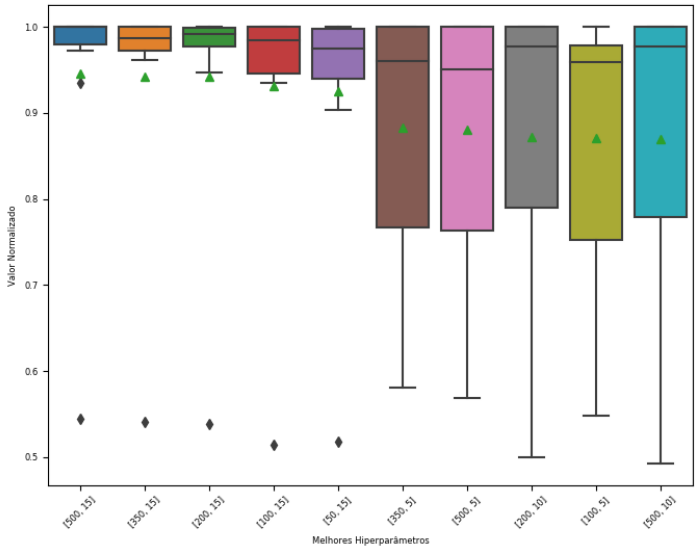


Figura 4: GRASP - Tempo de Execução

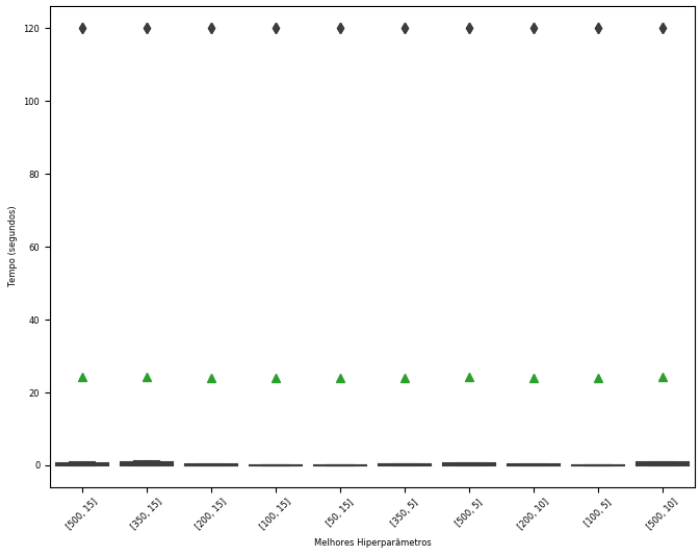


Figura 5: Resfriamento Simulado - Valores Normalizados

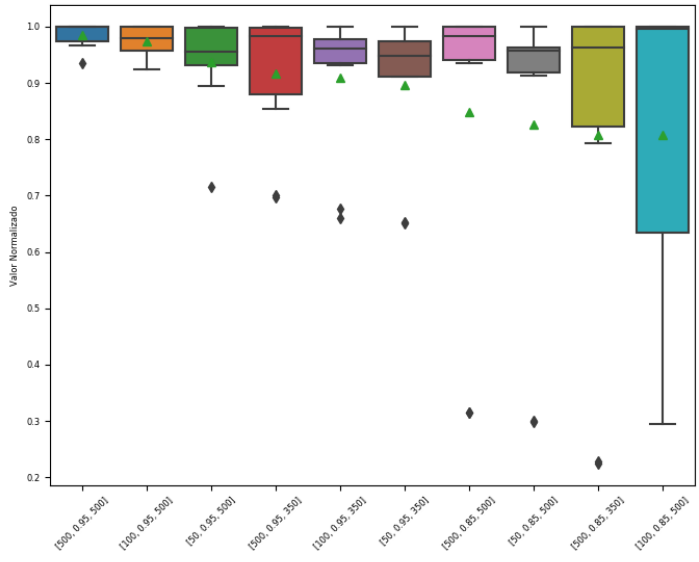


Figura 6: Resfriamento Simulado - Tempo de Execução

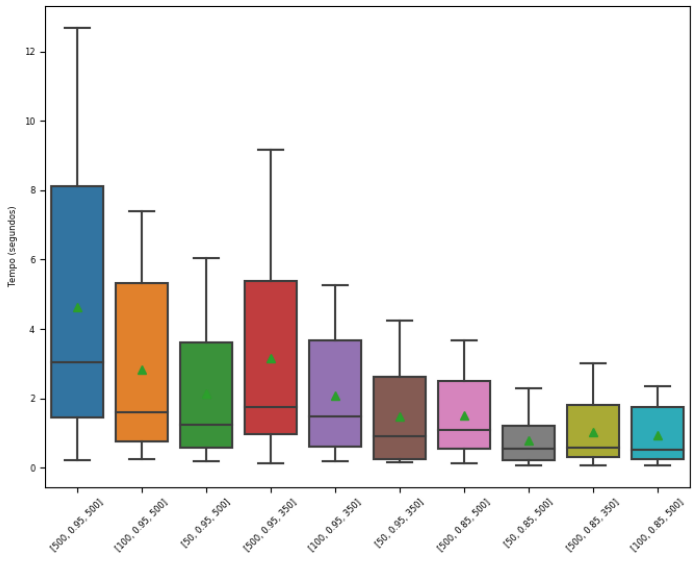


Figura 7: Algorítimo Genetico - Valores Normalizados

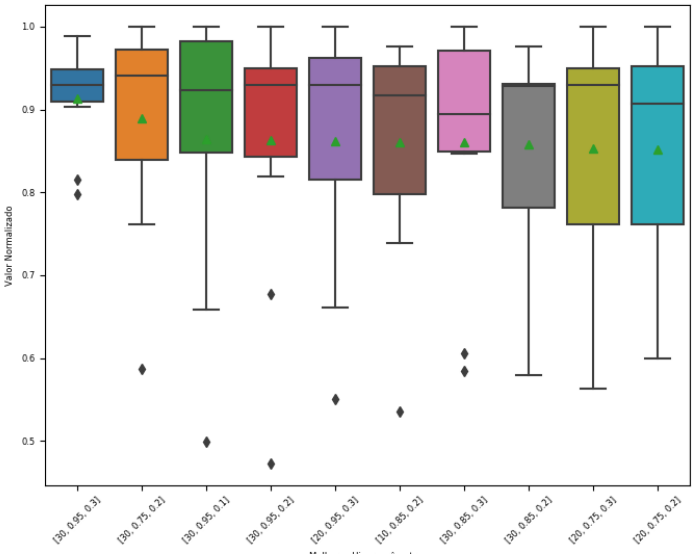
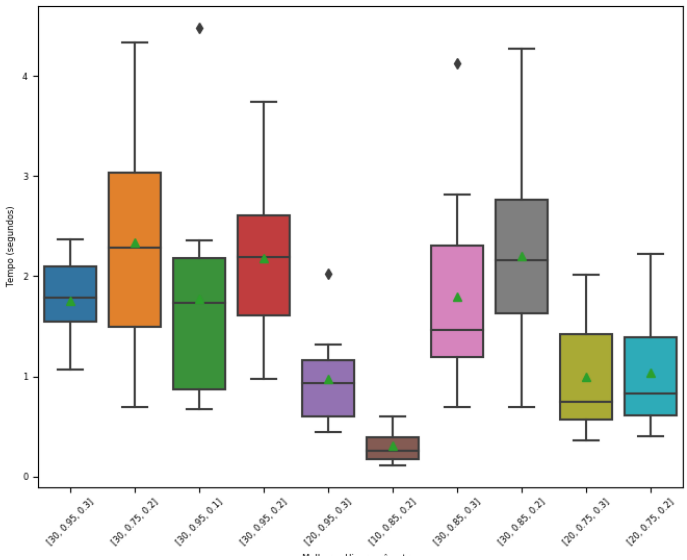


Figura 8: Algorítimo Genetico - Tempo de Execução



4.1.4. Análise dos Resultados Alcançados

Com os resultados mostrados nos gráficos podemos identificar várias características interessantes das diversas meta-heurísticas. A seguir analisaremos cada uma delas discutindo os resultados e apresentando algumas informações adicionais obtidas durante a execução.

A meta-heurística de Pesquisa por Feixe apresentou um comportamento singular com relação as demais. Para a maioria dos casos seu comportamento foi extremamente homogêneo, encontrando resultados muito parecidos em tempo praticamente idêntico (entre 0,9 e 1.0 segundos). Porém para os problemas P17 e P20, onde o tamanho da mochila é muito elevado e os tamanhos dos objetos relativamente muito pequenos, o tempo de execução explodiu, atingindo o limite de dois minutos com valores muito inferiores ao padrão. Isso ocorreu pois o algorítimo precisava seguir muitos ramos ao mesmo tempo de forma que o avanço em cada ramo foi muito lento nessas instâncias. É importante destacar que quando desconsiderados esses problemas o melhor hiperparâmetro encon-

trado foi $m = 100$ já que dividir mais o espaço de busca possibilitou examinar um maior número de soluções. Mas, considerando-os, esse grande número de ramos inviabilizou a descoberta de qualquer solução final dentro do limite de tempo. Dessa forma, a melhor opção foi utilizar de um número pequeno de ramos de forma a concentrar o poder computacional na maximização do resultado ao invés da abrangência.

O método GRASP apresentou graves problemas com relação ao tempo de execução. A primeira parte do algorítimo executa de forma quase idêntica a meta-heurística de Escalada porém mais lenta devido ao número de operações complexas utilizadas para realizar a escolha aleatória. Dessa forma, cada iteração do algorítimo necessita de tempo no mínimo igual a execução completa do algorítimo de Escalada. Dessa forma os supracitados problemas P17 e P20 estouraram o tempo limite de execução apresentando baixos resultados. Apesar disso podemos identificar outros aspectos interessantes do algorítimo. Observando o boxplot dos valores normalizados percebemos que os resultados variam muito quando o número de melhores elementos é 5 ou 10 e não apresenta bons resultados quando é 2 mas tornam-se estáveis com este parâmetro igual a 15. Isso concede ao algorítimo maior probabilidade de percorrer boas soluções já que amplia o espaço de busca. Naturalmente um elevado número de iterações provoca um efeito parecido pois aumenta o número de soluções observadas. Como essa meta-heurística utiliza de aleatoriedade ambos os fatores são muito importantes, permitindo explorar o espaço aleatório e consequentemente a probabilidade de encontrar melhores soluções.

Já é possível perceber que os problemas P17 e P20 mostraram-se cruciais para identificar as características das meta-heurísticas. Para o Resfriamento Simulado não foi diferente. Nesses dois problemas, apesar de não ser identificável devido a normalização dos resultados, os estados obtidos como resposta estavam muito longe de completar o espaço da mochila. Nem mesmo metade do tamanho total foi preenchido. Por outro lado o tempo de execução mal excedeu a faixa de 12 segundos, estando longe do limite. Isso deixa evidente que o avanço por iteração nesse algorítimo é muito pequeno se comparado aos demais. Claramente

a forma mais eficiente de utilizar essa meta-heurística é quando aplicada a um solução parcial próxima do ótimo. Mas mesmo nessa situação a busca tenderá a ocorrer próxima da solução dada, podendo encontrar máximos locais ao invés da solução ótima. Assim o melhor ajuste será justamente o que maximiza a distância percorrida pelo Resfriamento Simulado, ou seja, aumentando o número de iterações e a temperatura inicial como ocorreu na seleção. Sem duvidas todas essas características assemelham-se bastante ao processo físico no qual a meta-heurística foi baseada.

O Algorítimo Genético foi o que apresentou resultados mais heterogêneos. De fato essa meta-heurística baseia-se na constante alteração de seus indivíduos para explorar o espaço de busca, sendo o mais complexo para analisar seu comportamento e o que mais apresenta variação nos hiperparâmetros escolhidos. Por causa disso esse algorítimo foi testado diversas vezes para comparação dos hiperparâmetros escolhidos. O resultado apresentado foi o mais frequente dos encontrados e, assim como os demais, apresenta altos valores para todos os seus hiperparâmetros. Como ocorreu em outras das meta-heurísticas, esse ajuste possibilita um melhor aproveitamento do espaço de busca. Diferente do ocorrido com a Pesquisa por Feixe, a execução do Algorítimo Genético foi incrivelmente rápida, normalmente em menos de 5 segundos por problema. Isso demonstra que, apesar da inconstância da solução, essa meta-heurística pode facilmente iterar diversas vezes para buscar melhores soluções.

4.2. Etapa de Teste

Com os hiperparâmetros devidamente selecionados podemos realizar a etapa de testes. Nessa etapa cada meta-heurística executará uma vez para cada problema de teste selecionado. Assim como na etapa de treino 10 instâncias do problema foram selecionadas mas dessa vez cada algorítimo terá 5 minutos para executar, uma vez que o numero de execuções será consideravelmente menor.

4.2.1. Algoritmo de Teste

O seguinte algorítimo foi utilizado para a etapa de teste:

Algorithm 2 Algorítimo de Teste

```
1: função TESTE
2:     para cada meta-heurística faça
3:         para cada problema no conjunto de teste faça
4:             executar meta-heurística no problema
5:             armazenar valor da solução e tempo de execução
6:         fim para
7:         calcular média e desvio padrão dos resultados alcançados
8:         calcular média e desvio padrão dos tempos de execução
9:     fim para
10:    para cada problema no conjunto de teste faça
11:        normalizar os resultados alcançados pelas meta-heurísticas
12:    fim para
13:    calcular média e desvio padrão dos resultados normalizados
14:    gerar boxplot dos resultados alcançados pelas meta-heurísticas
15:    gerar boxplot dos tempos de execução das meta-heurísticas
16:    gerar tabela contendo média e desvio padrão absolutos e normalizados, e
       média e desvio padrão dos tempos de execução de todas as meta-heurísticas
17:    para cada problema no conjunto de teste faça
18:        fazer ranqueamento das meta-heurísticas segundo resultado absoluto
19:    fim para
20:    calcular média dos ranqueamentos das meta-heurísticas
21:    fazer ranqueamento das meta-heurísticas segundo as médias dos resulta-
       dos normalizados
22: fim função
```

4.2.2. Problemas de Teste

Os problemas utilizados na etapa de teste foram:

1. P2:

   Tamanho da Mochila: 192

   Lista de Itens ((vi,ti)):

   (1,3),(4,6),(5,7)

2. P5:

   Tamanho da Mochila: 287

   Lista de Itens ((vi,ti)):

   (1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,8),(8,9),(9,10)

3. P7:

   Tamanho da Mochila: 120

   Lista de Itens ((vi,ti)):

   (1,2),(2,3),(4,5),(5,10),(14,15),(13,20),(24,25),(29,30),(50,50)

4. P10:

   Tamanho da Mochila: 1240

   Lista de Itens ((vi,ti)):

   (1,2),(2,3),(3,5),(7,10),(10,15),(13,20),(24,25),(29,30),(50,50)

5. P12:

   Tamanho da Mochila: 104

   Lista de Itens ((vi,ti)):

   (25,26),(29,30),(49,50)

6. P13:

   Tamanho da Mochila: 138

   Lista de Itens ((vi,ti)):

   (1,3),(4,6),(5,7),(3,4),(2,6),(2,3),(6,8)

7. P15:

   Tamanho da Mochila: 13890

   Lista de Itens ((vi,ti)):

   (1,3),(4,6),(5,7),(3,4),(2,6),(2,3),(6,8),(1,2),(2,3),(3,5),(7,10),(10,15),(13,20),(24,25),(29,30),(50,50)

8. P16:

   Tamanho da Mochila: 13890

   Lista de Itens ((vi,ti)):

   (1,3),(4,6),(5,7),(3,4),(2,6),(2,3),(6,8),(1,2),(3,5),(7,10),(10,15),(13,20),(24,25),(29,37)

9. P18:

   Tamanho da Mochila: 190000

   Lista de Itens ((vi,ti)):

   (1,3),(4,6),(5,7)

10. P19:

    Tamanho da Mochila: 4567

    Lista de Itens ((vi,ti)):

    (1,3),(4,6),(5,7),(3,4),(2,6),(1,2),(3,5),(7,10),(10,15),(13,20),(15,20)

4.2.3. Apresentação e Análise dos Resultados

A seguir apresentaremos os resultados obtidos pelo algorítimo de teste. São eles a tabela contendo informações gerais das meta-heurísticas, os resultados dos ranqueamentos e os boxplots dos valores alcançados e tempo de execução das meta-heurísticas.

Tabela 1: Informações Gerais

Meta-heurística	Média Absoluta	Desvio Padrão Absoluto	Média Normalizada	Desvio Padrão Normalizado	Média do Tempo de Execução	Desvio Padrão do Tempo de Execução
Hill Climbing	16585.8	42153.3	0.994156	0.0104257	0.0329266	0.0793335
Beam Search	16587	42153	0.996216	0.00901768	0.540079	1.42405
Simulated Annealing	6671.6	11816.8	0.899053	0.220843	2.59157	2.03883
GRASP	14514.8	36971	0.925687	0.0643098	45.5957	81.1236
Genetic	410.6	421.799	0.524955	0.4327	1.89799	0.748277

Tabela 2: Média dos ranqueamentos dos Resultados Absolutos

Média das Colocações	Meta-heurística
1.7	Beam Search
2.1	Hill Climbing
2.8	Simulated Annealing
3.55	GRASP
4.85	Genetic

Tabela 3: Ranqueamento das Médias dos Resultados Normalizados

Colocação	Meta-heurística
1	Beam Search
2	Hill Climbing
3	GRASP
4	Simulated Annealing
5	Genetic

Figura 9: Valores Normalizado por Meta-heurística

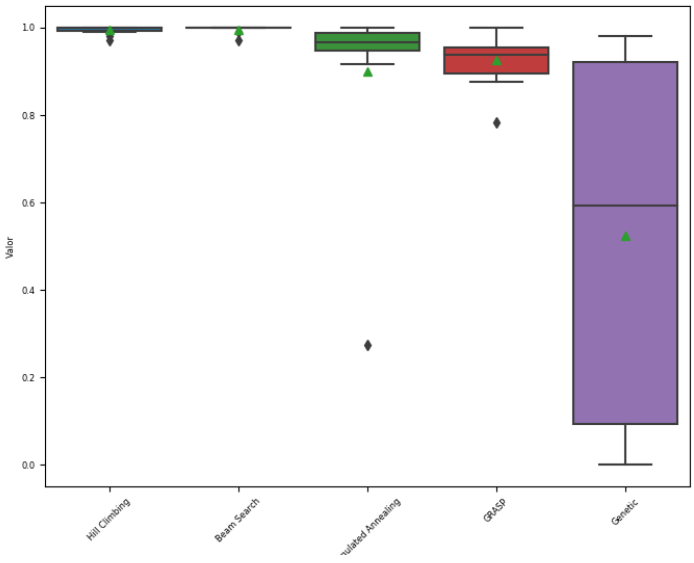
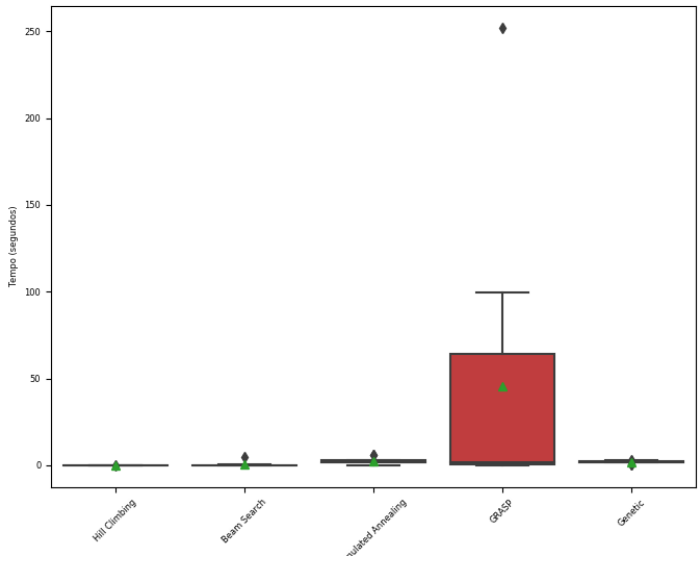


Figura 10: Tempos de Execução por Meta-heurística



O resultado dos testes foi compatível com o observado na etapa de treino. A meta-heurística GRASP novamente foi muito inferior em relação ao tempo de execução e o Algorítimo Genético apresentou os resultados mais variados. O Resfriamento Simulado apresentou resultados razoáveis mas novamente foi prejudicado por sua lentidão. Como já era esperado a Escalada e a Pesquisa por Feixe foram muito similares mas com uma pequena vantagem para essa última devido ao melhor aproveitamento do espaço de busca.

5. Conclusões

Neste relatório avaliamos o desempenho de algumas meta-heurísticas aplicadas com o problema da mochila com repetições e conseguimos identificar diversas características interessantes de todas elas. A diversidade quanto ao tempo de execução, qualidade dos resultados, aproveitamento do espaço de busca e outros fatores demostra que a escolha correta de uma meta-heurística que esteja em sincronia com as características do problema onde é aplicada pode fazer bastante diferença na experiencia da aplicação. Por exemplo, as meta-heurísticas de Escalada e Pesquisa por Feixe seriam aconselháveis para problemas que podem ser resolvidos eficientemente com abordagens gulosas enquanto o Algorítimo Genético pode ser mais adequado para problemas com muitas possíveis soluções distintas.