Relatório do Primeiro Trabalho Prático de Inteligência Artificial

Lorena B. Bassani[1]

Abstract

Este relatório sobre o primeiro trabalho prático da disciplina de Inteligência Artificial cursada no semestre de 2019/02 se dedica a realizar uma comparação experimental entre meta-heurísticas de busca. São comparadas, ao todo, cinco meta-heurísticas ensinadas na disciplina, sendo elas: Hill Climbing, Beam Search, Simulated Annealing, GRASP e Algoritmo Genético.

Keywords: Meta-heurística, Hill Climbing, Beam Search, Simulated Annealing, GRASP, Algoritmo Genético, Inteligência Artificial

1. Introdução

Este trabalho consiste em realizar uma comparação experimental entre um conjunto pré-definido de meta-heurísticas aplicadas ao problema da mochila. As meta-heurísticas escolhidas são: Hill Climbing, Beam Search, Simulated Annealing, Genetic Algorithm e GRASP.

O procedimento experimental será dividido em duas etapas: a primeira etapa consiste no ajuste de hiper-parâmetros das meta-heurísticas e a segunda etapa consiste na comparação das meta-heurísticas considerando apenas os valores dos hiper-parâmetros selecionados na primeira etapa. A meta-heurística Hill Climbing, que não possui hiper-parâmetros, só participará da segunda etapa da experimentação. Os problemas utilizados para a etapa de treino são necessariamente distintos dos problemas utilizados na etapa de teste.

O trabalho está estruturado de forma a apresentar o problema da mochila no

[1]Aluna de Ciência da Computação da Universidade Federal do Espírito Santo

tópico 2 e as meta-heurísticas em 3, sendo a Hill Climbing explicada em 3.1, a Beam Search em 3.2, a Simulated Annealing em 3.3, a GRASP em 3.4 e o Genetic Algorithm no tópico 3.5. Depois temos as explicações e resultados dos experimentos realizados, o que é apresentado no tópico 4, com o treino apresentado em 4.1 e os testes em 4.2. Por fim, a conclusão deste trabalho é apresentada no tópico 5.

2. Descrição do Problema da Mochila

2.1. Descrição geral do problema da mochila

O problema da mochila é um problema de otimização combinatória. O problema consiste de uma mochila com capacidade limitada e diversos itens com as suas utilidades (valores monetários, inclusive) e pesos conhecidos, e que devem ser transportados [1]. Este é um dos 21 problemas NP-Completos de Karp.
Por ser um problema NP-Completo, temos que o tempo de um algoritmo que encontre sempre a melhor resposta seria intratável. Ou seja, para instâncias muito grandes do problema, teríamos um tempo de execução tão grande que chegaria a ser inviável esperar tanto por uma resposta. Dessa forma, este trabalho propõe a solução aproximadamente boa para o problema através do uso de meta-heurísticas de busca.

2.2. Descrição das características particulares do problema específico utilizado nos experimentos

Para realizar os experimentos foi necessário modelar o problema de forma que fosse representável na linguagem escolhida, Python.
A modelagem realizada foi extremamente simples. Em um objeto do tipo Mochila, armazenamos as informações de tamanho máximo da mochila, na forma de um inteiro positivo, e as informações sobre valor e volume dos itens em uma lista de tuplas do formato (valor, volume). Nessa classe também implementamos a interface básica necessária para os algoritmos de busca agirem sobre ela, como por exemplo métodos de validação de estado, aptidão de estado, escolha

de melhor em uma lista de acordo com a aptidão, entre outros.

Um método que é interessante notar sobre a mochila é o método de aptidão, que retorna o valor total do estado considerando seus itens. Esse método é bastante simples de implementar para o problema da mochila dada a sua natureza de considerar melhor quanto maior o valor total dentro dela.

Um outro método também interessante de se notar é o método de geração de vizinhança. Ele procura gerar estados vizinhos a um dado estado. Ele tem um parâmetro *todaVizinhanca* que permite controlar se os estados gerados serão apenas estados melhores (adicionando um item) ou também estados piores (retirando um item).

3. Descrição dos Métodos Utilizados

3.1. Hill Climbing

Hill Climbing é uma meta-heurística baseada em soluções parciais. Seu procedimento é de Estratégia Gulosa. Os algoritmos de Hill Climbing realizam iterações onde a melhor solução conhecida é usada para gerar uma solução nova [2], sem backtracking. Ela não possui hiperparâmetros nem necessita de soluções iniciais para a busca.

3.2. Beam Search

Beam Search é uma meta-heurística baseada em soluções parciais.É um algoritmo de busca em amplitude, onde ao invés de utilizar apenas o melhor próximo estado, estuda as possibilidades de vários próximos estados expandindo-os. Pode-se dizer que ele é uma adaptação do método de Branch and Bound onde somente os nós mais promissores de cada nível da árvore de decisões (atribuições) são guardados na memória para serem visitados [3]. O número de estados estudados é passado como parâmetro, se tornando um hiperparâmetro.

3.3. Simulated Annealing

O Simulated Annealing é uma meta-heurística de busca local baseada em soluções completas. De acordo com Haeser e Ruggiero "o processo de otimização é realizado por níveis, simulando os níveis de temperatura no resfriamento." [4] Dessa forma, o algoritmo é baseado em processo físico de resfriamento de sólido superaquecido, onde durante o resfriamento podem ocorrer algumas etapas de medição de aquecimento, representado no algoritmo por uma chance de aceitar um estado pior que o atual. Quanto menor a "temperatura"menos o fenômeno de reaquecimento ocorre, ou seja, menor a chance de aceitarmos estados piores. Ele parte de um estado inicial recebido como parâmetro e conta com uma lista de hiperparâmetros :

- t : temperatura inicial
- a : taxa de queda de temperatura
- minT : temperatura mínima (critério de parada)
- numIter : quantidade de iterações por temperatura

3.4. GRASP

O GRASP é uma meta-heurística de busca local baseada em soluções completas. É um método iterativo probabilístico, onde cada iteração obtém uma solução independente do problema em estudo e é composta de duas fases.

1. Fase Construtiva : Determina uma solução a partir de uma função gulosa probabilística
2. Fase de Busca Local : Submeter a solução a um outro algoritmo de busca local.

A cada iteração do GRASP, é gerada uma solução por meio de uma heurística de construção, na qual é aplicada uma busca local. [5]
Seus hiperparametros são:

- m : elementos para escolher no construtor guloso de estados
- numIter : número de iterações total do algoritmo (critério de parada)

3.5. Genetic Algorithm

O Algoritmo Genético é uma meta-heurística de busca populacional por computação evolutiva baseada em soluções completas. São algoritmos probabilísticos que fornecem um mecanismo de busca paralela e adaptativa baseado no princípio de sobrevivência dos mais aptos e na reprodução [6], que trabalham em cima do conceito de gerações de uma população. Cada geração é composta por descendentes da geração anterior (recombinações ou mutações desses indivíduos) e são selecionados para continuar os melhores descendentes, com chances de alguns descendentes piores serem escolhidos no lugar de alguns melhores por fator aleatório. Ele não necessita de um estado inicial e seus hiperparametros são:

- maxIter : número máximo de iterações (critério de parada)
- tamanhoPop : tamanho da população
- maxSemMelhora : número máximo de iterações sem melhora de resposta (critério de parada)
- chanceCross : chance de ocorrer crossover
- chanceMutacao : chance de ocorrer mutação

Neste trabalho foi implementado um Algoritmo Genético com Função de Seleção por Roleta, onde a chance de um estado ser o selecionado para sobreviver é proporcional ao quão melhor ele é comparado aos outros; Função de crossover simples onde são selecionados uma quantidade de item de cada mochila e são invertidas entre si por um número arbitrário de vezes; e Função de Mutação simples que roda um número aleatório de vezes, onde é retirado ou adicionado 1 a quantidade de um item aleatório.

4. Descrição dos Experimentos Realizados

Esse trabalho é dividido em dois experimentos. O primeiro é o Treinamento de hiperparâmetros dos algoritmos de busca. Como cada algoritmo possui parâmetros de entrada, e precisamos encontrar os parâmetros que melhor se

ajustam ao problema específico da mochila. Esse treinamento é realizados em todos os algoritmos menos o Hill Climbing, que não possui hiperparâmetros, e é feito por busca em grade (*Grid Search*).

Já o Segundo experimento é o Teste dos Algoritmos, depois de treinados, para saber qual ou quais deles melhor se aplicam ao problema da Mochila da forma como foi modelado.

O primeiro experimento será descrito no sub-tópico 4.1, e o segundo no 4.2.

4.1. Treinamento

4.1.1. Descrição do Algoritmo de Treino

O Algoritmo de Treinamento realiza uma busca em grade, realizando todas as combinações de parâmetros que se deseja testar em um determinado conjunto de problemas de Treino. Esse algoritmo chega a conclusão do melhor conjunto de parâmetros encontrado, os tempos de execução e os resultados normalizados.

Algorithm 1 Algoritmo de Treinamento

Require: Conjunto de Problemas de Treinamento

Ensure: (Melhor conjunto de Parâmetros, Tempos de Execução, Resultados Absolutos e Normalizados)

```
for Cada m : metaheurística que necessita de ajuste de hiperparâmetros do
    for Cada g : combinação da busca em grade do
        for Cada p : problema do conjunto de treinamento do
            Resultados[p] ← Resultados[p] ∪ m(p, g) com Timeout de 2 min
        end for
    end for
    for Cada p : problema do conjunto de treinamento do
        ResultadosNormalizados ← normaliza(Resultados[p])
    end for
    for Cada g : combinação da busca em grade do
        m ← media(ResultadosNormalizados[g])
        if m é a maior dentre as médias then
            Utilizar a combinação g para o Teste
        end if
    end for
end for
Gerar Boxplots e Apresentar os valores de Hiperparâmetros escolhidos
```

4.1.2. Descrição dos Problemas de Treino

Os problemas de Treino são problemas necessariamente diferentes dos problemas presentes no conjunto de teste. Isso para assegurar que não estamos viciando (*Overfitting*) o nosso algoritmo no conjunto em que foi treinado e considerando esses resultados como bons.

O conjunto apresenta os seguintes problemas para treinamento (apresentados por nome : [(Valor do Item, Peso do Item)], tamanho máximo):

- "m1": [(1, 3), (4, 6), (5, 7)], 19

- "m3": [(1, 3), (4, 6), (5, 7), (3, 4)], 58
- "m4": [(1, 3), (4, 6), (5, 7), (3, 4), (8, 10), (4, 8), (3, 5), (6, 9)], 58
- "m6": [(1, 3), (4, 6), (5, 7), (3, 4), (8, 10), (4, 8), (3, 5), (6, 9), (2, 1)], 58
- "m8": [(1, 2), (2, 3), (4, 5), (5, 10), (14, 15), (15, 20), (24, 25), (29, 30), (50, 50)], 120
- "m9": [(1, 2), (2, 3), (3, 5), (7, 10), (10, 15), (13, 20), (24, 25), (29, 30), (50, 50)], 120
- "m11": [(24, 25), (29, 30), (50, 50)], 120
- "m14": [(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)], 138
- "m17": [(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)], 13890000
- "m20": [(1, 3), (4, 6), (5, 7), (3, 4), (2, 6), (1, 2), (3, 5), (7, 10), (10, 15), (13, 20), (15, 20)], 45678901

4.1.3. Descrição dos valores de hiperparâmetros utilizados na busca em grade de cada meta-heurística

Os valores de hiperparametros utilizados na busca em grade estão descritos abaixo para cada metaheurística que necessita de treinamento.

4.1.4. Beam Search

O Beam Search possui apenas um hiperparâmetro, sendo esse o número de estados que ele mantém durante a expansão de um estado, representado pelo parâmetro *nEstados*.

- nEstados (int) : 10, 25, 50, 100

4.1.5. Simulated Annealing

O Simulated Annealing possui quatro hiperparâmetros, sendo esses a temperatura inicial em que o algoritmo começa, representado pelo parâmetro t; a taxa de resfriamento α, representado pelo parâmetro a; o número de iterações para cada temperatura que o algoritmo passa, representado pelo parâmetro *numIter*; e o critério de parada de temperatura mínima que o algoritmo irá alcançar, representado pelo parâmetro *minT*.

- t (int) : 50, 90, 100, 250, 500
- a (float) : 0.7, 0.85, 0.9, 0.95, 0.97, 0.99
- numIter (int) : 50, 100, 200, 350, 500

Não foi de preocupação do trabalho especificar a temperatura mínima, que é um critério de parada do algoritmo. Dessa forma, utilizou-se o valor 1 como menor temperatura.

4.1.6. GRASP

O GRASP possui dois hiperparâmetros, sendo esses o número de melhores elementos para o construtor guloso, representado pelo parâmetro m; e o número de iterações que o algoritmo faz, representado por *numIter*.

- m (int) : 2, 5, 10, 15
- numIter (int) : 50, 100, 200, 350, 500

4.1.7. Genetic Algorithm

O algoritmo Genético possui cinco hiperparâmetros, sendo esses o número máximo de iterações como critério de parada, representado pelo parâmetro *maxIter*; o número máximo de gerações sem melhorias também como critério de parada, representado pelo parâmetro *maxSemMelhora*; o tamanho da população, representado pelo parâmetro *tamanhoPop*; a chance de ocorrer um crossover, representado pelo parâmetro *chanceCross*; e a chance de ocorrer uma mutação, representado pelo parâmetro *chanceMutacao*.

- maxIter (int) : 50, 100, 200, 350, 500
- maxSemMelhora (int) : 15, 20, 30, 50, 100
- tamanhoPop (int) : 10, 20, 30
- chanceCross (float) : 0.75, 0.85, 0.95
- chanceMutacao (float) : 0.1, 0.2, 0.3

4.1.8. Apresentação dos box plots e dos valores selecionados dos hiperparâmetros de cada meta-heurística

Neste tópico, apresenta-se os tempos e os dez melhores resultados de cada metaheurística durante a etapa de treinamento. No caso de haver menos combinações de hiperparâmetros que dez, são apresentados menos de dez resultados. Também apresentamos os valores de hiperparâmetros selecionados para a fase de teste.

4.1.9. Box plots dos hiperparâmetros e valores selecionados

Os hiperparâmetros selecionados foram:

Algoritmo Genético	"chanceCross": 0.95, "chanceMutacao": 0.3, "maxIter": 350, "maxSemMelhora": 50, "tamanhoPop": 10
Beam Search	"nEstados": 10
GRASP	"m": 2, "numIter": 100
Simulated Annealing	"a": 0.99, "minT": 1, "numIter": 500, "t": 500

Tabela 1: Hiperparâmetros Selecionados

Nos box plots apresentados nesta seção, quanto maior o índice, melhor o resultado apresentado.

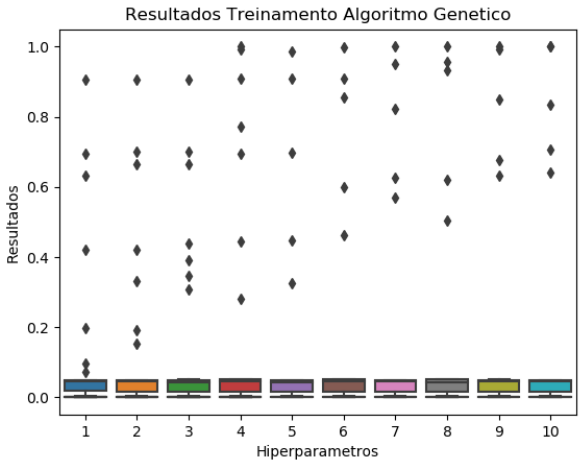

Figura 1: Melhores Resultados do Treinamento do Algoritmo Genético

1	'chanceCross': 0.75, 'chanceMutacao': 0.1, 'maxIter': 350, 'maxSemMelhora': 100, 'tamanhoPop': 20
2	'chanceCross': 0.75, 'chanceMutacao': 0.1, 'maxIter': 500, 'maxSemMelhora': 100, 'tamanhoPop': 20
3	'chanceCross': 0.75, 'chanceMutacao': 0.2, 'maxIter': 350, 'maxSemMelhora': 50, 'tamanhoPop': 10
4	'chanceCross': 0.75, 'chanceMutacao': 0.2, 'maxIter': 500, 'maxSemMelhora': 100, 'tamanhoPop': 20
5	'chanceCross': 0.75, 'chanceMutacao': 0.2, 'maxIter': 500, 'maxSemMelhora': 100, 'tamanhoPop': 30
6	'chanceCross': 0.75, 'chanceMutacao': 0.3, 'maxIter': 350, 'maxSemMelhora': 100, 'tamanhoPop': 10
7	'chanceCross': 0.85, 'chanceMutacao': 0.2, 'maxIter': 500, 'maxSemMelhora': 100, 'tamanhoPop': 10
8	'chanceCross': 0.85, 'chanceMutacao': 0.3, 'maxIter': 350, 'maxSemMelhora': 100, 'tamanhoPop': 10
9	'chanceCross': 0.95, 'chanceMutacao': 0.2, 'maxIter': 500, 'maxSemMelhora': 100, 'tamanhoPop': 10
10	'chanceCross': 0.95, 'chanceMutacao': 0.3, 'maxIter': 350, 'maxSemMelhora': 50, 'tamanhoPop': 10

Tabela 2: Melhores Resultados do Treinamento do Algoritmo Genético

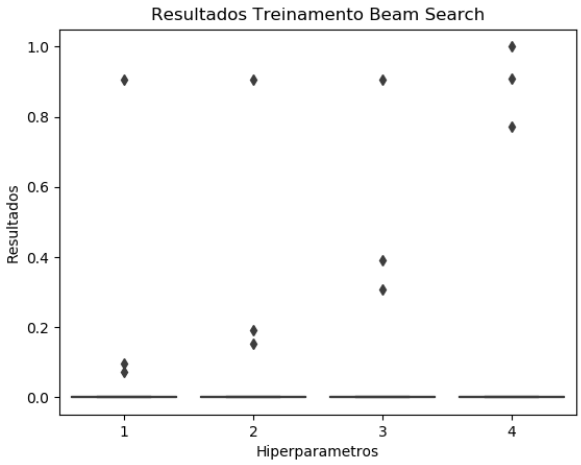

Figura 2: Melhores Resultados do Treinamento do Beam Search

1	'nEstados': 100
2	'nEstados': 50
3	'nEstados': 25
4	'nEstados': 10

Tabela 3: Melhores Resultados do Treinamento do Beam Search

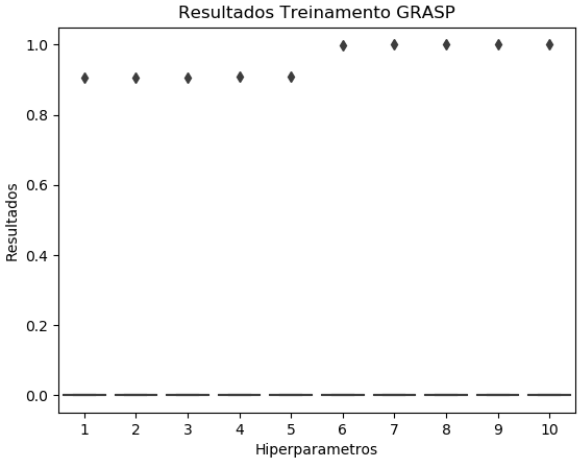

Figura 3: Melhores Resultados do Treinamento do GRASP

1	'm': 5, 'numIter': 350
2	'm': 5, 'numIter': 200
3	'm': 5, 'numIter': 500
4	'm': 5, 'numIter': 50
5	'm': 5, 'numIter': 100
6	'm': 2, 'numIter': 500
7	'm': 2, 'numIter': 50
8	'm': 2, 'numIter': 200
9	'm': 2, 'numIter': 350
10	'm': 2, 'numIter': 100

Tabela 4: Melhores Resultados do Treinamento do GRASP

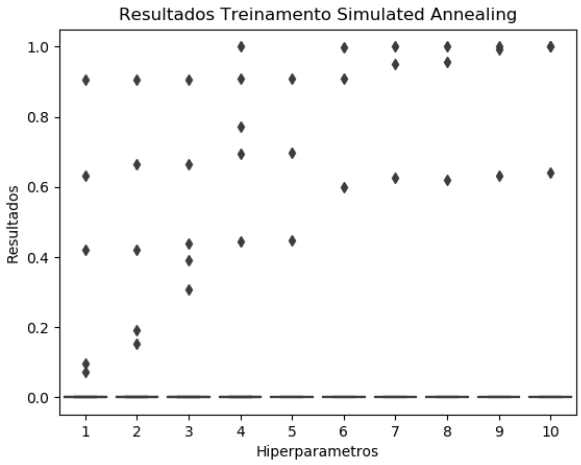

Figura 4: Melhores Resultados do Treinamento do Simulated Annealing

1	'a': 0.99, 'minT': 1, 'numIter': 350, 't': 50
2	'a': 0.99, 'minT': 1, 'numIter': 350, 't': 90
3	'a': 0.99, 'minT': 1, 'numIter': 350, 't': 100
4	'a': 0.99, 'minT': 1, 'numIter': 350, 't': 250
5	'a': 0.99, 'minT': 1, 'numIter': 350, 't': 500
6	'a': 0.99, 'minT': 1, 'numIter': 500, 't': 50
7	'a': 0.99, 'minT': 1, 'numIter': 500, 't': 90
8	'a': 0.99, 'minT': 1, 'numIter': 500, 't': 100
9	'a': 0.99, 'minT': 1, 'numIter': 500, 't': 250
10	'a': 0.99, 'minT': 1, 'numIter': 500, 't': 500

Tabela 5: Melhores Resultados do Treinamento do Simulated Annealing

4.1.10. Tempos de Treinamento

Nesse tópico apresentamos os Tempos de Treinamento de cada metaheurística.

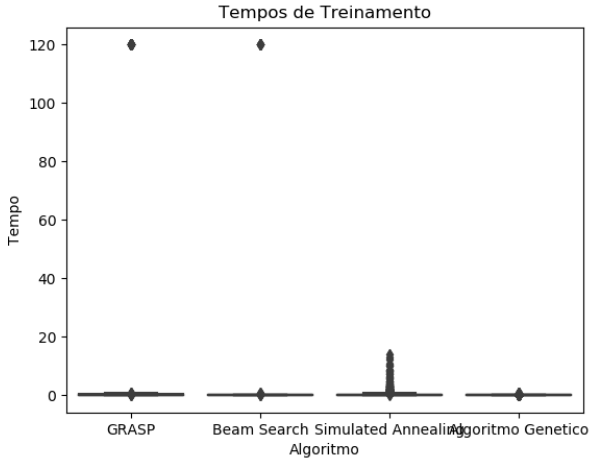

Figura 5: Tempos dos Treinamentos

4.1.11. Análise dos resultados alcançados

Alguns treinamentos foram especialmente demorados para realizar, com enfase no GRASP, que teve o maior índice de treinamentos alcançando o tempo máximo de execução sem termino.

Os treinamentos do Simulated Annealing e do Algoritmo Genético foram os mais rápidos para realizar.

4.2. Teste

4.2.1. Descrição do Algoritmo de Teste

O Algoritmo de Teste procura validar os resultados alcançados no treinamento, testando se os parâmetros encontrados efetivamente são bons para o problema, evitando assim que *overfittings* passem desapercebidos. Para isso, cada algoritmo rodará em todos os problemas do conjunto de teste descrito em 4.2.2 e retornará os resultados para os parâmetros escolhidos.

Algorithm 2 Algoritmo de Teste

Require: Conjunto de Problemas de Teste

Ensure: (Tempos de Execução, Resultados Absolutos e Normalizados)

```
for Cada m : metaheurística do
    for Cada p : problema do conjunto de teste do
        Resultados[p] ← Resultados[p] ∪ m(p)
    end for
    media[m] ← mediaAbsoluta(Resultados[m])
    desvio[m] ← desvioPadrão(Resultados[m])
end for
for Cada p : problema do conjunto de teste do
    ResultadosNormalizados[p] ← normaliza(Resultados[p])
    Ranquamento ← ranque(Resultados[p])
end for
for Cada m : metaheurística do
    mediaNormal[m] ← media(ResultadosNormalizados[m])
    desvioNormal[m] ← desvioPadrao(ResultadosNormalizados[m])
    mediaR[m] ← media(Ranqueamento[m])
end for
Gerar tabela com medias e desvios absolutos e normalizados dos Resultados
e absolutos dos tempos
Apresentar as metaheurísticas em ordem crescente de média de ranqueamento
```

4.2.2. Descrição dos Problemas de Teste

Os problemas de Teste são problemas necessariamente diferentes dos problemas presentes no conjunto de treino. Isso para assegurar que não estamos viciando (*Overfitting*) o nosso algoritmo no conjunto em que foi treinado e considerando esses resultados como bons.

O conjunto apresenta os seguintes problemas para treinamento (apresentados por nome : [(Valor do Item, Peso do Item)], tamanho máximo):

- "m2": [(1, 3), (4, 6), (5, 7)], 192

- "m5": [(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 10)], 287
- "m7": [(1, 2), (2, 3), (4, 5), (5, 10), (14, 15), (13, 20), (24, 25), (29, 30), (50, 50)], 120
- "m10": [(1, 2), (2, 3), (3, 5), (7, 10), (10, 15), (13, 20), (24, 25), (29, 30), (50, 50)], 1240
- "m12": [(25, 26), (29, 30), (49, 50)], 104
- "m13": [(1, 3), (4, 6), (5, 7), (3, 4), (2, 6), (2, 3), (6, 8)], 138
- "m15": [(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)], 13890
- "m16": [(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)], 13890
- "m18": [(1, 3), (4, 6), (5, 7)], 190000
- "m19": [(1, 3), (4, 6), (5, 7), (3, 4), (2, 6), (1, 2), (3, 5), (7, 10), (10, 15), (13, 20), (15, 20)], 4567

4.2.3. Apresentação da tabela contendo média e desvio padrão absolutos e normalizados, e média e desvio padrão dos tempos de execução de todas as meta-heurísticas

Algoritmo	**Media Absoluta**	**Desvio Absoluto**	**Media Normalizada**	**Desvio Normalizado**	**Media Tempo**	**Desvio Tempo**
Hill Climbing	16585.8	42153.2702	0.96962	0.0524	0.0273	0.0675
GRASP	16226.7	40841.8183	0.8030	0.3080	10.8523	26.4303
Simulated Annealing	15765.8	39414.3401	0.7486	0.3462	3.83641	2.1779
Beam Search	16586.9	42153.0765	0.9843	0.0341	0.2242	0.5175
Algoritmo Genetico	607.5	644.3043	0.1584	0.3409	0.0511	0.0388

Tabela 6: Tabela de Médias e Desvios Padrões

4.2.4. Apresentação de tabela contendo os ranqueamentos das meta-heurísticas segundo resultados absolutos para cada problema de teste e a média dos ranqueamentos

Nessa seção são apresentados o Ranqueamento das heurísticas por problema na tabela 7 e as médias dos Ranqueamentos na tabela 8, ambos por resultados absolutos.

Problema	R	Algoritmo	Média	Problema	R	Algoritmo	Média
	1.5	Hill Climbing	136		1.5	Hill Climbing	258
	1.5	Beam Search	136		1.5	Beam Search	258
m2	3.0	Simulated Annealing	135	m5	3.0	GRASP	256
	4.0	GRASP	134		4.0	Simulated Annealing	252
	5.0	Algoritmo Genetico	124		5.0	Algoritmo Genetico	242
	1.0	Hill Climbing	118		1.5	Hill Climbing	1236
	2.0	GRASP	117		1.5	Beam Search	1236
m7	2.0	Beam Search	117	m10	3.0	GRASP	1220
	4.0	Algoritmo Genetico	112		4.0	Simulated Annealing	1188
	5.0	Simulated Annealing	101		5.0	Algoritmo Genetico	1014
	1.5	Simulated Annealing	99		1.0	Beam Search	103
	1.5	Beam Search	99		2.0	Hill Climbing	102
m12	2.5	Hill Climbing	98	m13	3.0	GRASP	100
	2.5	Algoritmo Genetico	98		4.0	Simulated Annealing	98
	5.0	GRASP	83		5.0	Algoritmo Genetico	96
	1.5	Hill Climbing	13886		1.0	GRASP	11889
	1.5	Beam Search	13886		2.0	Simulated Annealing	11865
m15	3.0	GRASP	13696	m16	3.0	Beam Search	10895
	4.0	Simulated Annealing	13587		4.0	Hill Climbing	10885
	5.0	Algoritmo Genetico	1703		5.0	Algoritmo Genetico	1690
	1.5	Hill Climbing	135714		1.5	Hill Climbing	3425
	1.5	Beam Search	135714		1.5	Beam Search	3425
m18	3.0	GRASP	131547	m19	3.0	Simulated Annealing	3333
	4.0	Simulated Annealing	127000		4.0	GRASP	3225
	5.0	Algoritmo Genetico	355		5.0	Algoritmo Genetico	641

Tabela 7: Ranqueamento por Problema de Teste

Algoritmo	Média do Ranqueamento
Beam Search	1.65
Hill Climbing	1.85
GRASP	3.1
Simulated Annealing	3.45
Algoritmo Genetico	4.65

Tabela 8: Algoritmos por média de Ranqueamento

4.2.5. Apresentação dos box plots dos resultados absolutos e normalizados obtidos por cada metaheurística

Nesse tópico são apresentados os box plots dos resultados absolutos e normalizados de cada metaheurística.

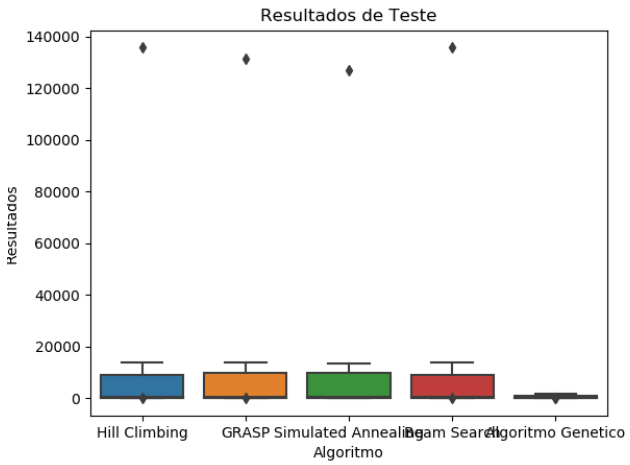

Figura 6: Resultados Absolutos dos Testes

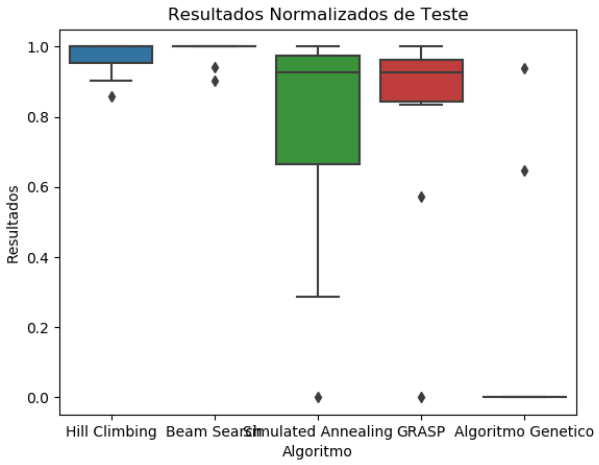

Figura 7: Resultados Normalizados dos Testes

4.2.6. Apresentação dos box plots dos tempos de execução obtidos por cada meta-heurística

Nesse tópico são apresentados os box plots dos tempos de execução de cada metaheurística.

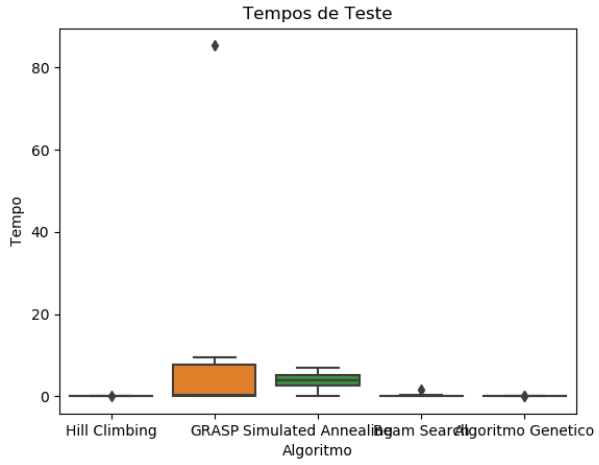

Figura 8: Tempos dos Testes

4.2.7. Análise dos resultados alcançados

É possível perceber que os resultados dos algoritmos Hill Climbing, GRASP e Simulated Annealing foram bem parecidos entre si, enquanto o Algoritmo Genético teve um desempenho relativamente baixo comparado a eles. Ainda assim, o tempo de execução do Algoritmo Genético foi o menor dentre eles.

A metaheurística que apresentou a melhor relação resultado e tempo entre as estudadas foi o Beam Search, que gerou resultados bons em tempos consideravelmente baixos, além de seu treinamento ser consideravelmente mais simples por ter apenas um hiperparâmetro.

5. Conclusões

Pelos testes, quatro dos cinco algoritmos obtiveram bons resultados. Porém, alguns tiveram treinamentos extremamente desgastantes, tomando muito tempo para obter resultados muito parecidos ou até não tão bons quanto de algoritmos mais simples. Esse caso pode ser reparado entre o GRASP e o Simulated Annealing, por exemplo. Enquanto o GRASP demorou muito para realizar os treinamentos, o Simulated Annealing foi mais rápido, e no final as metaheurísticas obtiveram resultados parecidos.

O destaque foi para o Beam Search, que sendo um algoritmo de implementação simples e apenas um hiperparâmetro, obteve resultados muito bons no teste. Seu treinamento foi simples e seus resultados foram muito satisfatórios.

Ainda foi possível reparar que o treinamento do Algoritmo Genético não obteve bons resultados. É possível que com outras combinações de hiperparâmetros o algoritmo tenha um bom desempenho. Novos treinamentos e testes ou talvez melhorias nos métodos internos como seleção ou crossover são objetivos para futuros trabalhos sobre algoritmos de busca.

Referências

[1] W. O. P. Rodrigues, C. C. de Souza, and J. F. dos Reis Neto, "O problema da mochila resolvido com algoritmos genéticos usando a ferramenta ms excel 2003,"

[2] N. Grando *et al.*, "Desenvolvimento de um método baseado em algoritmos de inteligência artificial para ajustar aparelhos auditivos," 2016.

[3] C. M. Ribeiro, A. T. Azevedo, and F. LIMA, "Resolução do problema de carregamento e descarregamento de contêineres em terminais portuários via beam search," *XLII Simpósio*, 2010.

[4] G. Haeser and M. G. Ruggiero, "Aspectos teóricos de simulated annealing e um algoritmo duas fases em otimização global," *Trends in Applied and Computational Mathematics*, vol. 9, no. 3, pp. 395–404, 2008.

[5] A. T. Lopes, V. M. L. Schulz, and G. R. Mauri, "Grasp com path relinking para o problema de alocação de berços," *Pesquisa Operacional para o Desenvolvimento*, vol. 3, no. 3, pp. 218–229, 2011.

[6] M. A. C. Pacheco *et al.*, "Algoritmos genéticos: princípios e aplicações," *ICA: Laboratório de Inteligência Computacional Aplicada. Departamento de Engenharia Elétrica. Pontifícia Universidade Católica do Rio de Janeiro. Fonte desconhecida*, p. 28, 1999.