Universidade Federal
de São João del-Rei

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
HEURÍSTICAS E METAHEURISTICAS

Problemas da Mochila Binária e do Caxeiro Viajante com Simulated Annealing e Busca Tabu

Gustavo Henriques da Cunha

São João del-Rei
2024

Lista de Figuras

Sumário

1 INTRODUÇÃO

Este é um trabalho prático da disciplina de Heurísticas e Metaheuristicas no curso de Ciência da Computação na UFSJ.

1.1 Objetivo

Neste trabalho, temos como objetivo aprender a construir e testar os algoritmos *Simulated Annealing* e Busca Tabu, buscando resolver o problemas da mochila binária e do caxeiro viajante.

Além disso, focamos na análise da calibragem dos parâmetros, buscando melhorar a eficiência dos algoritmos em cada problema.

2 OS PROBLEMAS

Neste trabalho iremos trabalhar com dois problemas muito famosos na Computação, exemplos clássicos para o teste de heurísticas e metaheuristicas, que são o problema da mochila binária e o problema do caxeiro viajante.

2.1 PROBLEMA DA MOCHILA BINÁRIA

O problema da mochila bináaria, ou problema da mochila 0/1, é um problema de otimização combinatória que consiste na maximização da utilidade dos itens levados(v) em uma mochila de capacidade (c), cada um dos n itens candidatos com peso em (p).

Podemos formular o problema formalmente, como, dado dois vetores p e v de n posições e um número c, buscamos encontrar um subconjunto $\mathcal{X}$ de $\{0, 1, ..., n-1\}$ que maximize $v(\mathcal{X})$, a função de avaliação da solução, sobre uma restrição $\sum p(\mathcal{X}) \leq c$.

2.2 PROBLEMA DO CAXEIRO VIAJANTE

O problema do caixeiro-viajante é um dos exemplos de problemas de otimização combinatória mais famosos que encontramos na computação. Por ser, em sua natureza, um problema difícil de resolver, pertencente à classe NP-Difícil, mas fácil de explicar e ilustrar, ele se tornou um problema comum para testarmos diferentes algoritmos de aproximação, nos quais buscamos encontrar uma solução boa em tempo polinomial.

Dado um grafo $G = (V, A)$ onde temos n vértices V que representam as cidades e arestas A que ligam essas cidades com certo peso que representa a distância entre elas, precisamos encontrar um circuito de menor distância possível que comece em uma cidade, passando por todas apenas uma vez e então retornando a cidade inicial.

3 OS ALGORITMOS

Nesta seção introduzimos os algoritmos usados no trabalho.

3.1 SIMULATED ANNEALING

O *SimulatedAnnealing* é uma metaheuristica baseada em processos metalúrgicos de aquecimento e esfriamento de ligas metálicas. O algoritmo busca, de forma probabilística, gerar diferentes vizinhos para uma solução inicial e escolher-los, dados dois critérios.

No primeiro critério, é analisado a qualidade da solução. Se a solução for melhor do que a solução atual ela é aceita.

No segundo critério, onde a solução é pior que a atual, usamos uma função probabilística para decidirmos se a escolhemos ou não. Essa função é dada pela equação (1):

$$p = \frac{1}{1 + e^{\frac{f(s) - f(s')}{T}}} \tag{1}$$

Onde $f(s)$ é a função objetivo aplicada a solução atual, $f(s')$ é a função objetivo aplicada a nova solução, e T é a temperatura, que é um parâmetro definido pelo usuário. Essa função, ao permitir a escolha de soluções piores, permite a fuuga de um possível melhor local, podendo posteriormente melhorar a melhor solução encontrada.

Dada a solução inicial, é gerado um vizingo aleatório, usado a função da equação (1) para verificarmos a aceitação ou caso sua avaliação seja melhor, e se aceito, repetimos o processo com essa nova solução. Repetimos o processo por um número de iterações especificado pelo usuário.

Depois do número de iterações se esgotar, a temperatura é atualizada a partir da sua multiplicação for uma taxa α, também especificada pelo usuário, que representa o fator de resfriamento.

Com a temperatura atualizada, o ciclo de iterações de escolha de vizinhos é repetido.

Quando a temperatura fica muito próxima de zero o algoritmo pode ser interrompido, retornando a melhor solução encontrada.

3.2 BUSCA TABU

A Busca Tabu é um algoritmo determinístico que trabalha a solução buscando fugir de ótimos locais.

Ela funciona ao guardar uma lista tabu que contém os movimentos necessários para chegar em todas as soluções vizinhas de uma solução atual. Inicialmente essa lista fica vazia e, dado uma solução inicial, geraremos todos os seus vizinhos mais próximos e iremos mudar para aquele com melhor valor da função objetivo do problema.

Quando mudamos de solução, marcamos o passo que gastamos para chegar nela na lista tabu com um número especificado pelo usuário que representa o tamanho da lista. Quando esse valor é diferente de zero, significa que o paço é tabu e não pode ser usado.

Para a nova solução, repetimos o paço de gerar seus vizinhos, mas ao invés de apenas escolhermos o vizinho com melhor valor, devemos verificar a lista tabu para ver se o paço usado para chegar nele não é tabu. Caso seja, deve ser selecionado o próximo passo de melhor valor que não seja tabu.

Dependendo do valor de tamanho da lista especificado, todos os movimentos podem ser tabu. Nesse caso, devemos escolher o movimento mais antigo da lista e fazer ele não tabu.

Os passos do algoritmo são repetidos até o critério de parada definido for atingido.

4 ABORDAGEM DO PROBLEMA

Nesta seção abordaremos a forma em que cada problema foi implementado pelos diferentes algoritmos.

4.1 Problema da Mochila Binária

Nas soluções implementadas, os items a serem adicionados na mochila e a capacidade máxima da mochila são lidos de um arquivo e armazendados, no caso do item, em um array de uma *struct* item, criada para armazenar o valor e o peso de cada item, e no caso da capacidade em uma variável. A solução é dos item a serem adicionados são armazenados

em uma string binária onde 0 mostra que o item não está presente na mochila e 1 o contrário.

4.1.1 Simulated Annealing

Para resolver o problema utilizando o *SumulatedAnnealing*, primeiro devem ser definidos três parâmetros principais:

- O número máximo de iterações, onde o algoritmo irá buscar por vizinhos da solução;
- A temperatura inicial, que diz a probabilidade de uma solução ruim ser escolhida;
- A taxa de resfriamento, que diz o quanto a temperatura irá diminuir após as iterações de busca por vizinho.

Outro parâmetro importante que foi implementado é o número máximo de iterações sem melhora da solução. Assim, o critério de parada é quando a temperatura fica muito próxima de zero ou quando o número de iterações sem melhora passa o limite.

Começamos o algoritmo de uma solução inicial onde apenas é adicionado um item aleatório a mochila. Depois, até o número de iterações ser passado, é gerado vizinhos e esses vizinhos podem ser aceitos se forem melhor que a solução atual ou se passarem pela função de Cálculo de Probabilidade.

4.1.2 Busca Tabu

Para resolver o problema utilizando a Busca Tabu, primeiro devem ser definidos dois parâmetros principais:

- O número máximo de iterações sem melhora da melhor solução;
- O tamanho da lista tabu.

O critério de parada é quando o número de iterações sem melhora passa o limite.

Para gerar a solução inicial usamos um algoritmo guloso baseado no custo beneficio.

Dado a solução inicial, até o critério de parada ser atingindo, é gerado todos os vizinhos da solução e é selecionado o melhor vizinho disponível na lista tabu. Sempre que um vizinho é escolhido, a lista tabu é atualizada.

4.2 Problema do Caxeiro Viajante

Em nosso problema, armazenaremos o grafo em uma matriz, lendo os dados de um arquivo contendo a coordenada cartesiana das cidades, sendo a distância entre elas calculada no código. Note que, para esse problema, só consideramos grafos completos.

Cada solução é um vetor de inteiros, onde cada interio representa uma cidade.

Para avaliarmos cada solução calculamos a distância do caminho.

4.2.1 Simulated Annealing

Para resolver o problema utilizando o *SumulatedAnnealing*, como para resolver o problema da mochila, primeiro devem ser definidos os mesmos três parâmetros principais:

- O número máximo de iterações, onde o algoritmo irá buscar por vizinhos da solução;
- A temperatura inicial, que diz a probabilidade de uma solução ruim ser escolhida;
- A taxa de resfriamento, que diz o quanto a temperatura irá diminuir após as iterações de busca por vizinho.

Começamos com uma solução inicial que é uma permutação aleatória das cidades.

O funcionamento do algoritmo será também o mesmo de como é no problema da mochila, onde só mudamos as estruturas de dados e a forma de avaliação da solução, nesse caso a distância. O critério de parada é quando a temperatura chega próxima a zero.

4.2.2 Busca Tabu

Como no problema da mochila, para resolver o problema do caxeiro viajante utilizando a Busca Tabu, primeiro devem ser definidos dois parâmetros principais:

- O número máximo de iterações sem melhora da melhor solução;
- O tamanho da matriz tabu.

Note que no caso do pcv, ao invés de uma lista tabu temos uma matriz tabu, pois armazenamos nela a aresta que devemos mudar para chegar em uma solução vizinha.

Para gerar uma solução inicial, utilizamos uma heurística gulosa que constrói, a partir de uma cidade aleatória, uma solução buscando sempre a cidade mais próxima.

As estruturas de dados usadas são as mesmas do *SimulatedAnnealing* e o funcionamento da busca funciona da mesma forma que no problema da mochila.

5 ANÁLISE DE DESEMPENHO

Para analisarmos o desempenho dos algoritmos, devemos ajustar os seus parâmetros e observarmos os valores das soluções encontradas bem como o tempo gasto para realizar a execução. Com isso, utilizando um *script* para facilitar múltiplas execuções, montamos tabelas e gráficos para vizualizar os efeitos dos parâmetros utilizados na solução.

5.1 Simulated Annealing

Como o *Simulated Annealing* é um algoritmo probabilístico, usamos dez execuções para cada parâmetro testado e colocamos em tabelas da figura, que mostram ordenadamente os resultados dos testes com as médias e os devios do tempo de execução e da avaliação da solução.

5.1.1 Problema do Caxeiro Viajante

Utilizando a entrada do arquivo "tsp_51", colocamos na figura 1.

Iterações	Temperatura	Resfriamento	Média	Desvio	Media Tempo	Desvio Tempo
350	800.0	0.99	464.81	5.86	11.07	0.52
250	800.0	0.99	467.15	9.00	8.21	0.39
350	1500.0	0.99	473.53	10.82	12.51	0.59
350	1000.0	0.99	476.13	22.53	11.54	0.36
250	1000.0	0.99	480.95	15.44	8.49	0.33
250	1000.0	0.98	484.47	14.97	5.76	0.11
150	800.0	0.99	487.77	18.69	5.07	0.21
350	2000.0	0.97	489.26	12.86	7.23	0.35
350	1500.0	0.98	490.39	10.96	8.51	0.41
350	1000.0	0.98	491.76	25.76	8.36	0.40
350	5000.0	0.97	499.26	18.39	7.40	0.18

Figura 1: PCV com SA - Tabela com as execuções

Podemos ver que o algoritmo conseguiu bons resultados com um número pequeno de iterações e uma temperatura relativamente pequena se observarmos a taxa de resfriamento alta.

Podemos também observar o impacto do número de iterações no tempo de execução nas duas primeiras entradas da tabela, onde o único parâmetro alterado foi o número de iterações. Assim é importante acharmos um meio-termo entre o número de execuções e

o tempo, pois um aumento nesse parâmetro pode sim melhorar os resultados mas nem sempre isso irá valer a pena.

Usando os parâmetros que geraram a melhor média de distância na solução, construímos dois gráficos que mostram o progresso, em dez execuções, das soluções geradas em cada geração.

Assim, a figura 2 mostra o progresso da melhor solução encontrada a cada vez em que a temperatura é atualizada. Já a figura 3 mostra o progresso de cada solução escolhida, seja por ser melhor que a solução atual ou seja pela função probabilística.

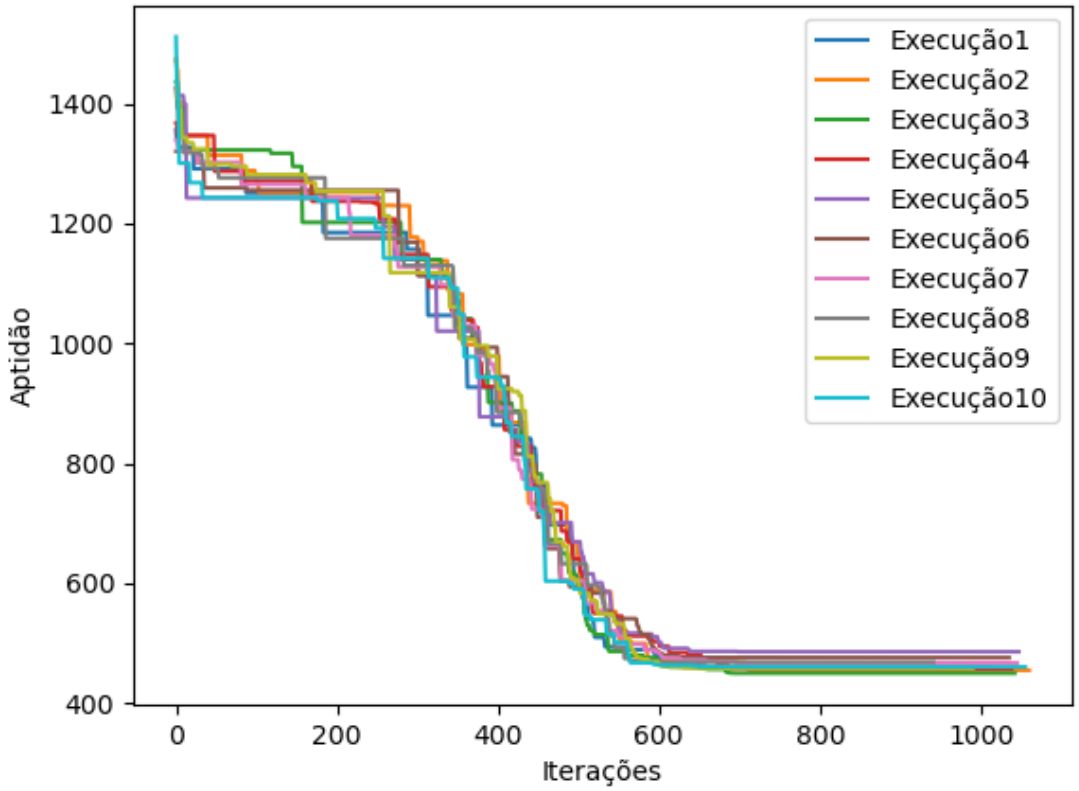

Figura 2: PCV com SA - Gráfico do melhor na atualização da Temperatura

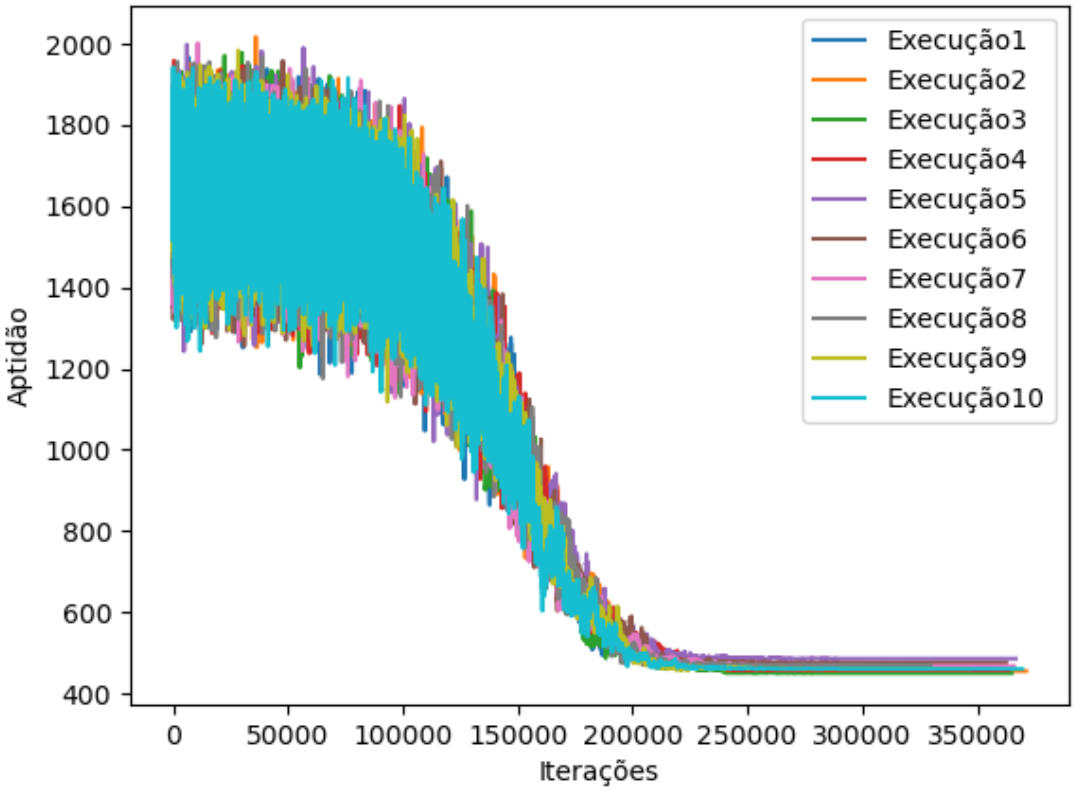

Figura 3: PCV com SA - Gráfico com cada solução escolhida

Em ambos os casos podemos observar como acontece a convergência da melhor solução. Em especial a figura 3 mostra a como o algoritmo seleciona soluções piores do que a obtida através da função probabilística, mas que isso ainda leva a uma melhora geral nas soluções encontradas conforme passam as iterações.

5.1.2 Problema da Mochila Binária

Utilizando dos mesmos métodos usados no pcv, também montamos uma tabela para os vários conjuntos de parâmetros usados no *Simulated Annealing* para a mochila 0/1, mostrada na figura 4, com a entrada "mochila_100_1000 _1".

Iterações	Temperatura	Resfriamento	Média	Desvio	Media Tempo	Desvio Tempo
8000	500.0	0.99	8891.22	113.50	121.91	27.78
5000	800.0	0.99	8808.00	176.81	54.84	13.06
5000	10000.0	0.99	8752.33	136.36	60.35	21.66
1000	8000.0	0.99	8584.44	210.99	14.05	2.97
1000	800.0	0.94	8575.22	189.34	8.85	0.70
1000	8000.0	0.93	8485.00	195.55	8.36	0.38
1000	800.0	0.96	8481.89	186.36	11.64	2.89
500	5000.0	0.96	8468.89	159.39	6.51	1.11
350	800.0	0.999	8257.44	199.66	4.44	2.23
350	1500.0	0.99	8236.56	254.22	4.75	1.33
350	800.0	0.99	8219.67	228.41	3.98	1.06

Figura 4: Mochila com SA - Tabela com as execuções

Podemos observar que quanto mais aumentamos o número, conseguimos melhores resultados mas ficamos com o tempo de execução maior. Para tentar amenizar o tempo, foi diminuido o valor da temperatura, mas a influência no tempo ainda foi pouca. Assim, para problemas maiores seria de maior interesse que o tempo não crescesse muito, então o resultado acabaria sendo prejudicado.

Com o conjunto de parâmetros que gerou melhor resultado, montamos um gráfico na figura 5 com dez execuções mostrando o progresso da melhor solução cada vez que a temperatura é atualizada.

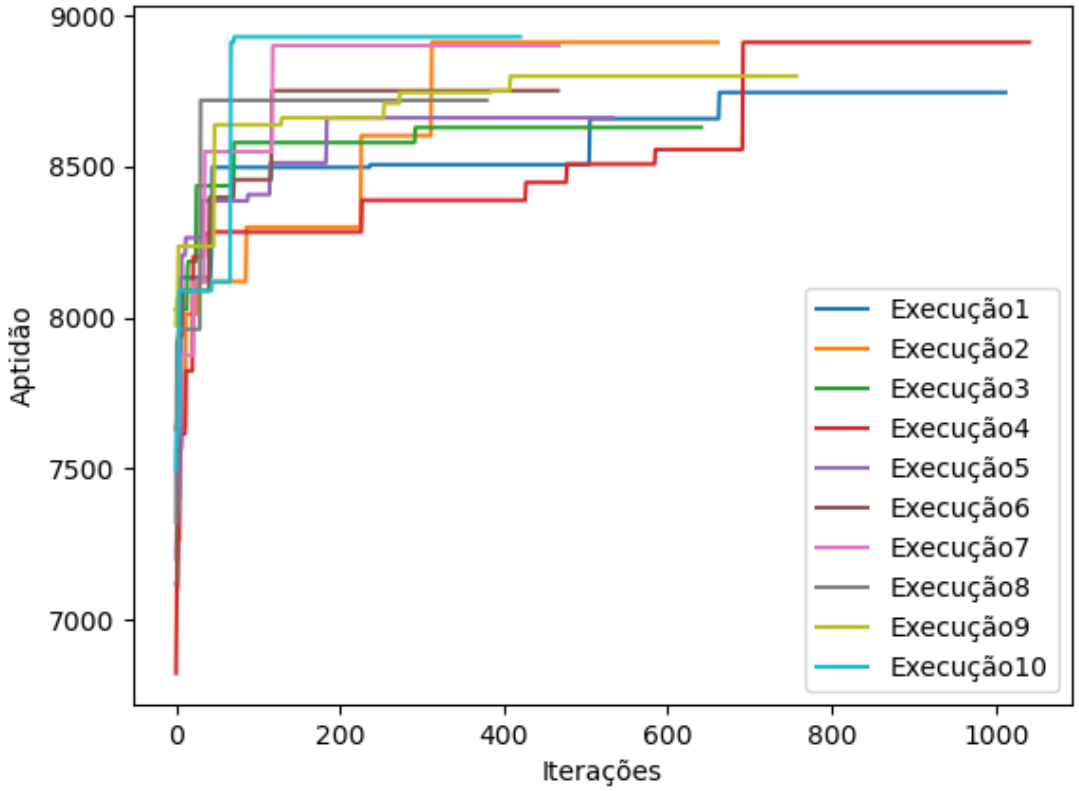

Figura 5: Mochila com SA - Gráfico do melhor na atualização da Temperatura

Podemos observar que diferentemente do pcv, onde em todas as execuções a convergência ocorria de maneira semelhante, neste caso ele ocorre em iterações bem diferentes para cada caso.

Podemos observar no gráfico também o critério de parada utilizado de número máximo de iterações sem melhorias. Com ele, as iterações que convergiram cedo pouparam tempo de execução.

5.2 Busca Tabu

Como a Busca Tabu é um algoritmo determinístico, não é necessário a execução com os mesmos parâmetros várias vezes, pois dado uma solução inicial, o algoritmo sempre resultará no mesmo resultado. Mas como utilizamos heurísticas gulosas para gerar soluções iniciais em ambos os problemas, e tendo que essas heurísticas utilizam de um fator probabilístico para construirem suas soluções, a busca tabu estará trabalhando com diferentes soluções iniciais em cada execução, o que irá gerar diferentes resultados.

Assim, para testar esse algoritmo e avaliar os parâmetros a serem usadaos, fizemos várias execuções com cada conjunto de parâmetros e ordenamos cada solução encontrada em uma tabela, mostrando a trinta melhores, e comparando o tempo de execução entre elas.

5.2.1 Problema do Caxeiro Viajante

Para o pcv, utilizando mais uma vez a entrada "tsp_51", montamos uma tabela com os resultados obtidos na figura 6.

Valor	Tempo	Iterações	Tamanho Lista
462.79071	0.024141	200	1
462.79071	0.024304	200	1
462.79071	0.024451	200	1
462.79071	0.02458	100	1
462.79071	0.024649	200	1
462.79071	0.025049	100	1
462.79071	0.025675	500	2
462.79071	0.026098	250	2
462.79071	0.026155	500	2
462.79071	0.026407	250	2
462.79071	0.026764	100	2
462.79071	0.027466	100	2
462.79071	0.028739	100	1
462.79071	0.028804	500	2
462.79071	0.029728	100	2
462.79071	0.030946	100	2
462.79071	0.040489	500	10
462.79071	0.040748	200	10
462.79071	0.040825	200	10
462.79071	0.041292	500	10
462.79071	0.041364	100	10
462.79071	0.04779	100	13
462.79071	0.050245	100	15
462.79071	0.050514	100	15
462.79071	0.050655	100	13
462.79071	0.05084	200	15
462.79071	0.050983	100	15
462.79071	0.051381	500	15
462.79071	0.051679	100	15
462.79071	0.052076	200	10

Figura 6: PCV com BT - Tabela com os resultados

Podemos observar que todos os itens na tabela, os trinta melhores dos testes gerados, tem o mesmo valor, possivelmente o ótimo global da solução. Isso mostra que a busca tabu consegue com frequência gerar uma boa solução mesmo com diferentes parâmetros.

Também podemos notar que os primeiros itens da lista todos têm um número baixo na lista tabu, o que permitem a eles um melhor tempo de execução.

5.2.2 Problema da Mochila Binária

Para o problema da mochila, também usamos mais uma vez o arquivo "mochila_100_1000 _1"como entrada e colocamos o resultado na tabela da figura 7.

Valor	Tempo	Iterações	Tamanho Lista
9147.0	0.000189	25	1
9147.0	0.000275	50	2
9147.0	0.000279	50	2
9147.0	0.000362	50	3
9147.0	0.00041	50	3
9147.0	0.00045	25	2
9147.0	0.000451	50	2
9147.0	0.000459	25	2
9147.0	0.000462	50	2
9147.0	0.000466	25	2
9147.0	0.00047	50	2
9147.0	0.000472	50	2
9147.0	0.000485	25	2
9147.0	0.000493	25	2
9147.0	0.000523	50	4
9147.0	0.000524	50	2
9147.0	0.000539	15	5
9147.0	0.000544	50	3
9147.0	0.000546	50	2
9147.0	0.000549	15	5
9147.0	0.000553	25	2
9147.0	0.000553	50	2
9147.0	0.000553	50	3
9147.0	0.000556	50	3
9147.0	0.000558	50	2
9147.0	0.00056	50	3
9147.0	0.000561	25	2
9147.0	0.000568	50	2
9147.0	0.000569	25	2
9147.0	0.000576	50	3

Figura 7: Mochila com BT - Tabela com os resultados

Mais uma vez todas as entradas possuem o mesmo valor, neste caso, conhecidamente o ótimo global da entrada. Assim, também faremos a comparação pelo tempo de execução e, mais uma vez, os conjuntos de parâmetros com menor tamanho da lista tabu tiveram vantajem no quesito tempo. Observe também que variámos bastante o tamanho do número de iterações e mesmo com ele baixo o algoritmo ainda assim consegue achar o melhor global dependendo da solução inicial, o que mostra sua eficiência para conseguir fugir de um ótimo local.

6 CONCLUSÃO

Ao compararmos os resultados obtidos em ambos os algoritmos, percebemos uma vantagem da busca tabu sobre o *Simulated Annealing*.

Mas veja que isso depende muito da solução inicial utilizada, onde houve casos nos testes que a busca tabu ficou muito longe do ótimo global pois teve uma solução inicial ruim.

No caso do *Simulated Annealing* independentemente da solução inicial, na maioria dos casos, ele consegue melhorá-la bastante, mas acaba gastando uma boa quantidade de

tempo. Em especial no problema da mochila, onde em poucos momentos chegou próximo da solução ótima e quando chegou gastou muito tempo. No caso do pcv ele funciona bem, e poderia ser tranquilamente escolhido sobre a busca tabu.

No mais vale a pena notar que ambos os algoritmos não se excluem. Na verdade é uma opção viável executar o *Simulated Annealing* para gerar uma solução próxima da ótima global e depois usar a busca tabu para tentar melhorá-la, mesmo que um pouco.