UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS GRADUAÇÂO EM CIÊNCIAS E TECNOLOGIA DA COMPUTAÇÃO

PCO117 - Introdução a Meta-Heurística

Prof. Dr. Rafael Francisco dos Santos

Prof. Dr. Sandro Carvalho Izidoro

Meta-Heurística

Busca-Tabu Aplicada ao Problema da Mochila 0/1

Autor

Breno de Oliveira Renó
brenooliveirareno@unifei.edu.br

09 de Maio de 2022

Sumário

1 Introdução

A Busca Tabu foi proposta por Fred Glover em 1986. Surgiu como uma técnica para guiar uma heurística de Busca Local na exploração do espaço de soluções além do ótimo local, através do uso de estruturas de memória [Talbi, 2012].

A Busca Tabu se comporta de maneira semelhante a um algoritmo de Busca Local, mas aceita soluções em que uma melhoria não acontece, com o objetivo de "escapar"do ótimo local. Desta forma a Busca Tabu consegue percorrer mais espaços de busca, enquanto não encontra seu critério de parada definido.

Para evitar ciclos, a Busca Tabu descarta os vizinhos que foram visitados anteriormente, memorizando a trajetória de busca recente. O algoritmo Tabu gerencia uma memória das soluções ou movimentos aplicados recentemente, que é chamada de lista Tabu. Esta lista Tabu constitui a memória de curto prazo. A cada iteração do algoritmo, essa memória de curto prazo é atualizada [Talbi, 2012].

O problema da Mochila (no original *knapsack problem*) é um problema de optimização combinatória. Toda a família de Problemas da Mochila requer que um subconjunto de itens sejam escolhidos, de tal forma que o somatório dos seus valores seja maximizado sem exceder a capacidade da mochila.

No problema da Mochila 0/1 (0/1 *Knapsack Problem*), cada ítem pode ser escolhido no máximo uma vez, enquanto que no problema da Mochila Limitado (*Bounded Knapsack Problem*) temos uma quantidade limitada para cada tipo de ítem. O problema da Mochila com Múltipla Escolha (*Multiple-choice Knapsack Problem*) ocorre quando os itens devem ser escolhidos de classes disjuntas, e se várias Mochilas são preenchidas simultaneamente temos o problema da Mochila Múltiplo (*Multiple Knapsack Problem*). A forma mais geral é o problema da Mochila com multi restrições (*Multi-constrained Knapsack Problem*) o qual é basicamente um problema de Programação Inteira Geral com Coeficientes Positivos [Caldas and Ziviani, 2004].

1.1 Objetivos

Tendo como base o contexto apresentado, o objetivo deste trabalho é implementar a metaheurística Busca-Tabu aplicada ao problema da Mochila 0/1, utilizando a base de dados disponível em [Ortega, 2022]. Para o desenvolvimento do trabalho foram definidas as seguintes exigências:

- O tamanho da vizinhança de uma solução, deve ser no mínimo d ≤ 2.
- Deve ser implementado a lista tabu de forma eficiente, proposta no material da disciplina.
- Deve ser implementado o critério de aspiração por objetivo global e *default*.

2 Código

Uma vez definidos os objetivos do trabalho, o algoritmo Tabu foi implementado em linguagem C, utilizando como base o código de busca local apresentado na disciplina. O tamanho da lista Tabu foi definido como 7, sendo esse o "número mágico"proposto por alguns pesquisadores presentes na literatura.

A implementação da lista foi feita através de uma estrutura de dados, que controla tanto os elementos considerados Tabu, quanto a posição onde a lista se encontra. Essa estrutura de dados pode ser vista na Figura 1.

```
//Definições da Lista
#define tamLista 7
struct listaTabu
{
    float Elementos[tamLista]; // Lista de Itens Proibidos
    int posicao;               // Posição em que a Lista se Encontra
};

//Utilizando a Lista de maneira Global
listaTabu Lista;
```

Figura 1: Código - Definição da Lista

A lista Tabu foi implementada para funcionar de maneira cíclica. Desta forma, quando a variável de controle apontar para o último elemento da lista, representado pelo elemento 6 no vetor, a próxima posição apontada será a primeira posição do vetor, como pode ser visto na Figura 2. De acordo com a entrada de novos valores na lista, os valores mais antigos são substituídos, sendo esse um critério para uma solução sair da lista Tabu.

```
if (Lista.posicao == tamLista - 1 || Lista.posicao == -1) {
    Lista.posicao = 0;
    Lista.Elementos[Lista.posicao] = solucao.valor;
} else {
    Lista.posicao++;
    Lista.Elementos[Lista.posicao] = solucao.valor;
}
```

Figura 2: Código - Manejando a Lista

Dois critérios de parada foram definidos para a execução do algoritmo, inicialmente foi definido como critério de parada o alcance do valor ótimo da instância, afinal, não faz sentido continuar a execução do algoritmo caso o melhor valor possível já tenha sido encontrado. Posteriormente foi definido o segundo critério de parada como a ocorrência de 10 iterações do algoritmo sem melhoria na solução global como pode ser visto na Figura 3. O segundo critério foi implementado para evitar a possibilidade do algoritmo encontrar um *loop* infinito ou o tempo de execução se tornar muito grande mas sem obter resultados interessantes.

```
//Critério de parada: 10 iterações sem melhoras na solução global
int criterioParada = 10;
int falhas = 0;

while (falhas <= criterioParada)
{
    ...
```

Figura 3: Código - Critério de Parada

A função responsável pela a avaliação de vizinhança foi dividida em duas novas funções, ”avaliaVizinhanca”e ”avaliaVizinhancaTamanhoDois”, vistas na Figura 4. Além disso, um novo parâmetro booleano foi adicionado, que aponta se a busca é Tabu ou se busca é apenas local. Essas mudanças foram feitas com o objetivo de facilitar os testes e facilitar também a comparação dos resultados.

```
//Distância = 1
avaliaVizinhanca(solucao, mochila, itens, true);
//Distância = 2
avaliaVizinhancaTamanhoDois(solucao, mochila, itens, true);
```

Figura 4: Código - Tamanho da Vizinhança

.

A Figura 5 apresenta o conteúdo de um arquivo gerado após a execução do algoritmo. O arquivo está organizado de maneira a mostrar cada iteração e o valor alcançado, separado por ponto e vírgula. A iteração marcada como -1 representa o valor ótimo da instância e a última iteração, representada por -2, representa a quantidade de execuções que ocorreram sem resultar em uma melhora a solução global.

```
Iteração;Valor
-1;295
0;85
1;233
2;293
3;290
4;295
-2;1
```

Figura 5: Exemplo de Saida

.

Ainda referente a Figura 5, é possível ver a solução inicial (iteração número 0) partindo do valor 85, sendo melhorada para 233, e depois para 293. Nesse ponto o algoritmo ”dá um passo atrás”para o valor 290 em busca de encontrar uma melhor solução, e encontra. O resultado encontrado foi 295, que é justamente o ótimo da instância, e sendo esse um critério de parada, o algoritmo se encerra.

3 Experimentos

Uma vez implementado, o algoritmo foi executado nas 10 instâncias *“Low-dimensional”* e em 10 instâncias *“Large-scale”* como proposto inicialmente. Com a execução do algoritmo foi possível coletar informações sobre o número de iterações realizadas, o número de falhas (execuções que não geraram melhoria na solução global), o melhor valor encontrado e o valor ótimo para cada instância. Para a realização dos experimentos, o tamanho da vizinhança foi definido como igual a 2, desta forma utilizando a função “avaliaVizinhancaTamanhoDois”.

3.1 *Low-dimensional*

Os dados provenientes da aplicação do algoritmo nas instâncias *Low-Dimensional* podem ser vistos na Tabela 1.

Tabela 1: Instancias *Low Dimensional*

Instância de Dados	**Iterações**	**Falhas**	**Melhor**	**Ótimo**
f1_l-d_kp_10_269	4	1	295	295
f2_l-d_kp_20_878	9	0	1024	1024
f3_l-d_kp_4_20	3	1	35	35
f4_l-d_kp_4_11	1	0	23	23
f5_l-d_kp_15_375	25	10	481.069	481.069
f6_l-d_kp_10_60	2	0	52	52
f7_l-d_kp_7_50	1	0	107	107
f8_l-d_kp_23_10000	6	0	9767	9767
f9_l-d_kp_5_80	2	0	130	130
f10_l-d_kp_20_879	8	0	1025	1025

3.2 *Large-scale*

Os dados provenientes da aplicação do algoritmo nas instâncias *Large-scale* podem ser vistos na Tabela 2.

Tabela 2: Instancias *Large Scale*

Instância de Dados	**Iterações**	**Falhas**	**Melhor**	**Ótimo**
knapPI_1_100_1000_1	10	0	9147	9147
knapPI_1_200_1000_1	27	10	9738	11238
knapPI_1_500_1000_1	37	10	24218	28857
knapPI_1_1000_1000_1	50	10	41466	54503
knapPI_1_2000_1000_1	70	10	72395	110625
knapPI_1_5000_1000_1	129	10	143036	276457
knapPI_1_10000_1000_1	226	10	274534	563647
knapPI_2_500_1000_1	27	10	3343	4566
knapPI_3_100_1000_1	12	10	1496	2397
knapPI_3_1000_1000_1	11	10	6290	14390

3.3 Gráficos

Posteriormente à aplicação dos algoritmos, foram construídos gráficos que mostram a evolução do algoritmo aplicado em 6 diferentes instâncias escolhidas. Um dos gráficos construídos em questão está disponível para acesso público através da plataforma *Tableau Public* e pode ser encontrado no link: https://tabsoft.co/39HVOCD. Uma vez construídos os gráficos, foi possível traçar algumas conclusões relacionadas ao comportamento do algoritmo.

4 Discussão dos Resultados

As informações presentes na Tabela 1 mostram que, para as instâncias *Low-dimensional* o resultado ótimo sempre foi alcançado. A Figura 6 apresenta o gráfico construído com os dados referentes a instância f1_l-d_kp_10_269. É possível ver que na terceira iteração o algoritmo dá "um passo atrás"antes de encontrar o ótimo da instância, que é 295.

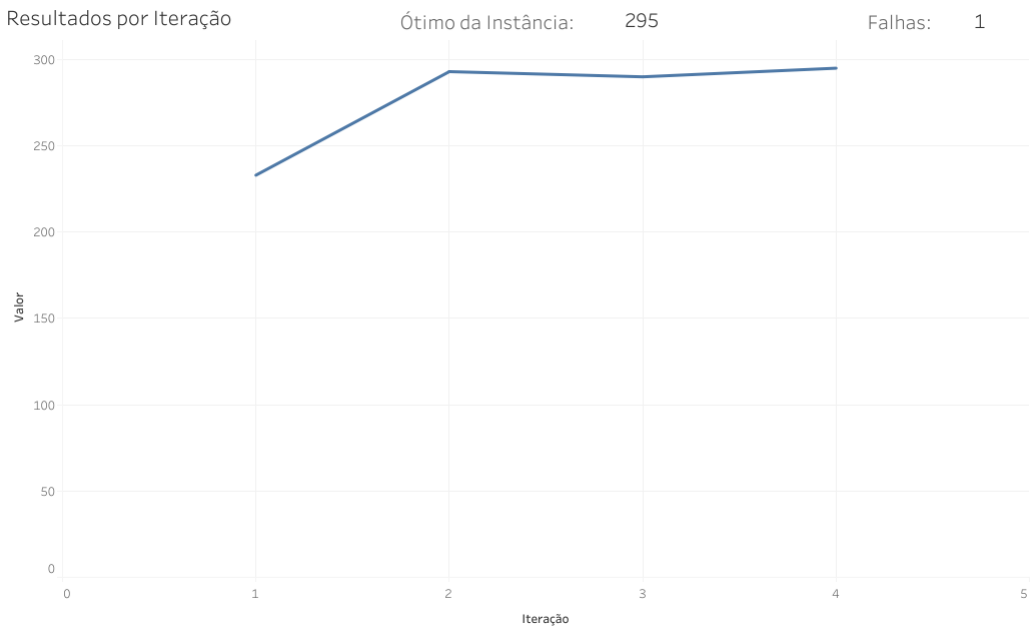

Figura 6: Gráfico - f1_l-d_kp_10_269

.

A Figura 7 mostra o gráfico construído com os dados referentes a instância f2_l-d_kp_20_878. Esse foi o comportamento mais comum nessa família de instâncias, onde o valor ótimo foi encontrado sem ocorrer uma falha no processo.

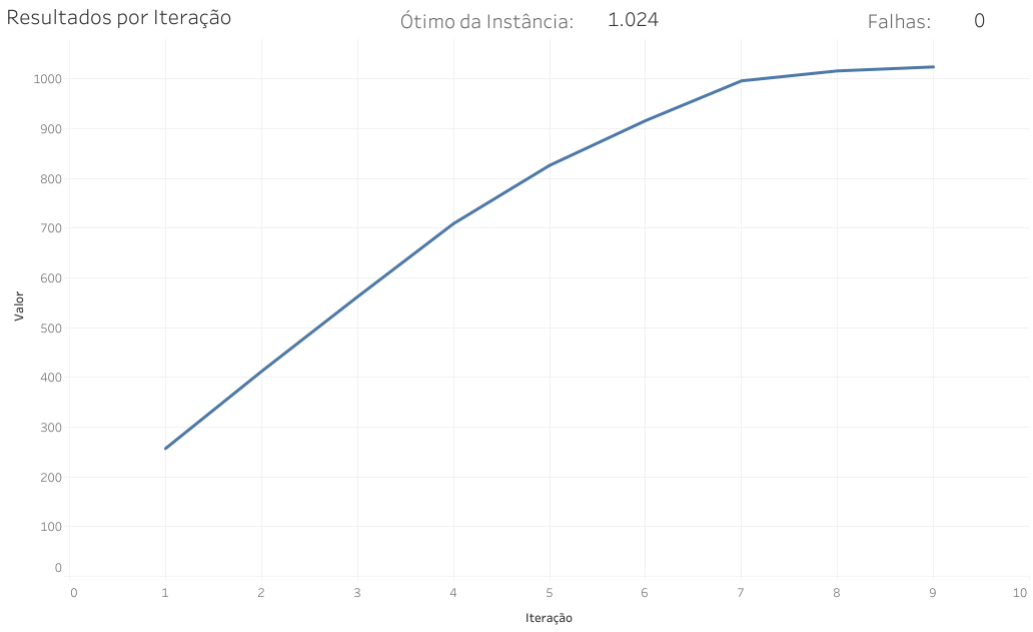

Figura 7: Gráfico - f2_l-d_kp_20_878

.

A Figura 8 mostra o gráfico construído com os dados referentes a instância f3_l-d_kp_4_20. Esse é mais um exemplo em que o funcionamento da lista Tabu fica claro, onde na segunda iteração é possível ver uma queda no gráfico, mas que depois, leva a uma evolução até o ótimo da instância.

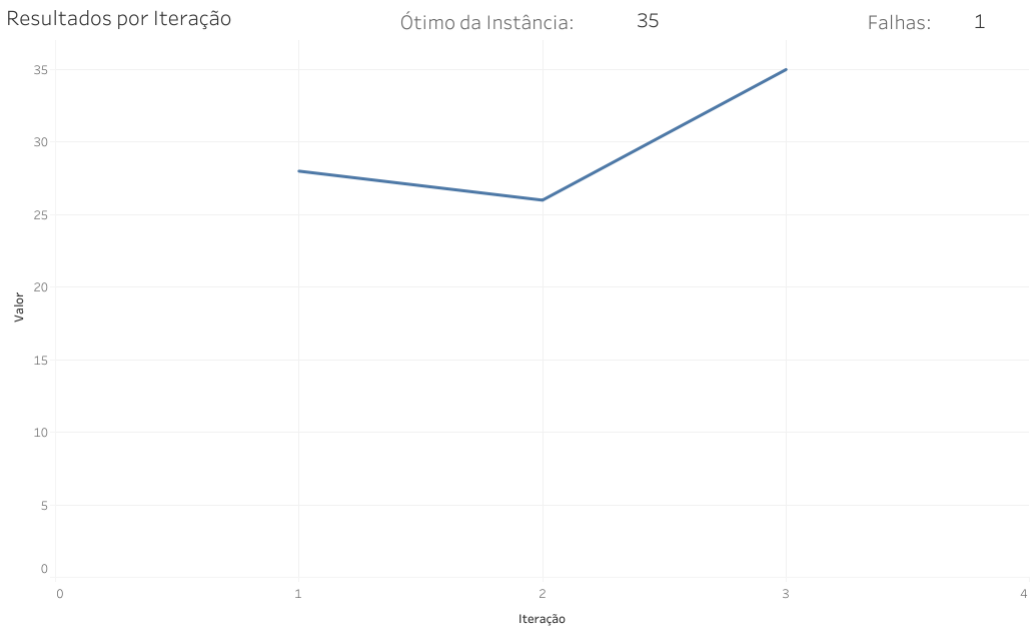

Figura 8: Gráfico - f3_l-d_kp_4_20

.

A Figura 9 apresenta o primeiro dos gráficos construídos em cima das instâncias *"Large-scale"*. Neste caso, o comportamento da instância knapPI_1_1000_1000_1 foi igual a maior parte das instâncias deste tipo, onde o algoritmo alcançou um resultado alto, porém, após as 10 falhas definidas como critério de parada ele se encerrou antes de encontrar o ótimo da instância.

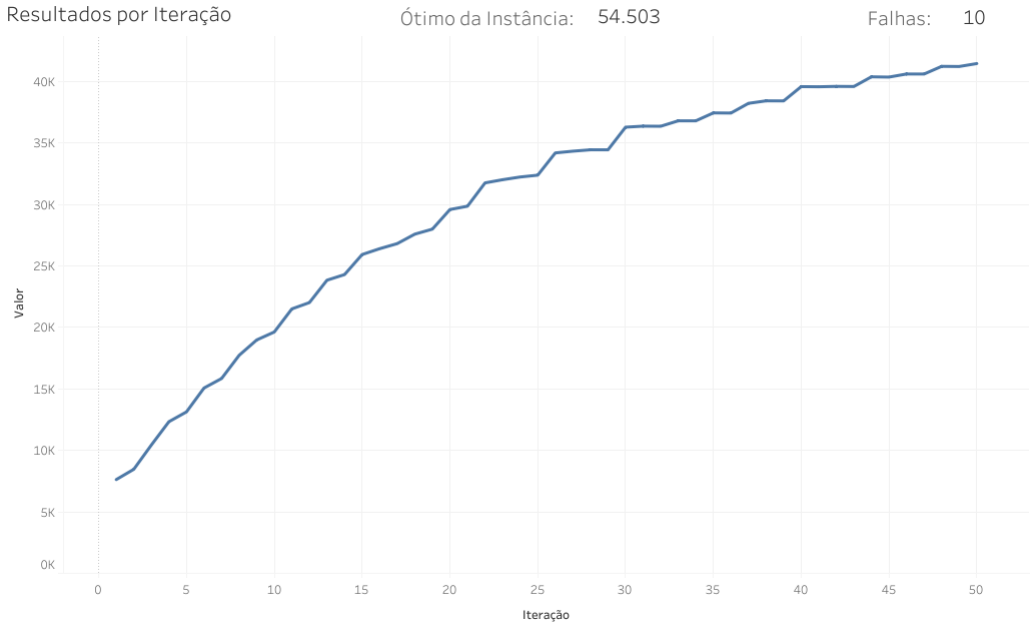

Figura 9: Gráfico - knapPI_1_1000_1000_1

.

A Figura 10 apresenta a instância knapPI_1_2000_1000_1, que também atingiu o critério de parada definido após 10 falhas, porém, alcançando no processo, valores mais altos que uma busca local permitiria.

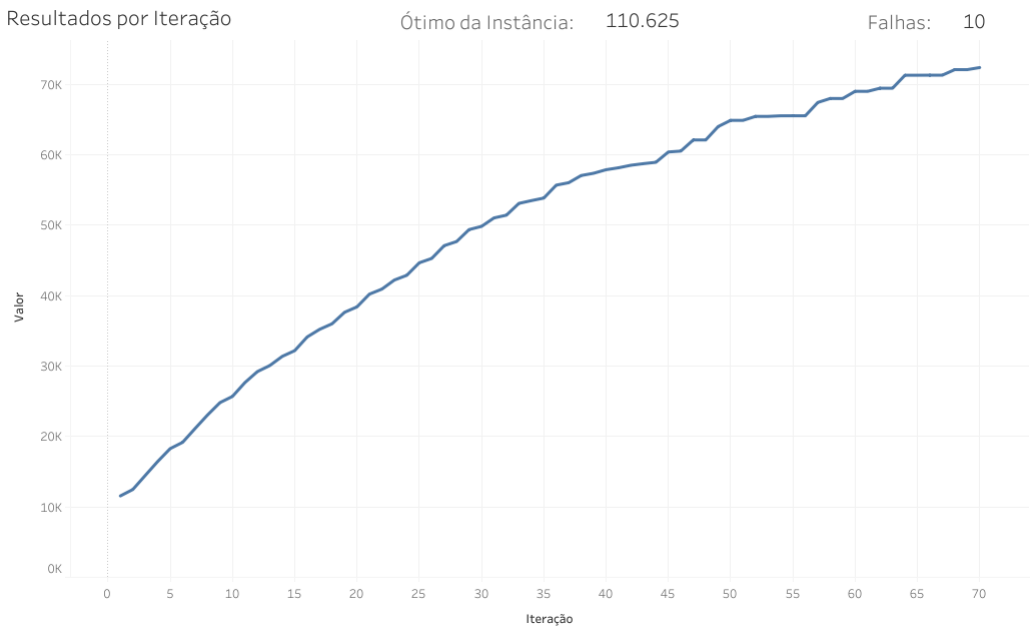

Figura 10: Gráfico - knapPI_1_2000_1000_1

.

A Figura 11 apresenta a instância knapPI_1_5000_1000_1, que por sua vez, também seguiu o mesmo padrão. É válido apontar que em instâncias maiores de dados é difícil encontrar os pontos exatos onde o comportamento do algoritmo se mostra presente, mas com o gráfico interativo e com os dados textuais em mão essa tarefa se torna mais fácil.

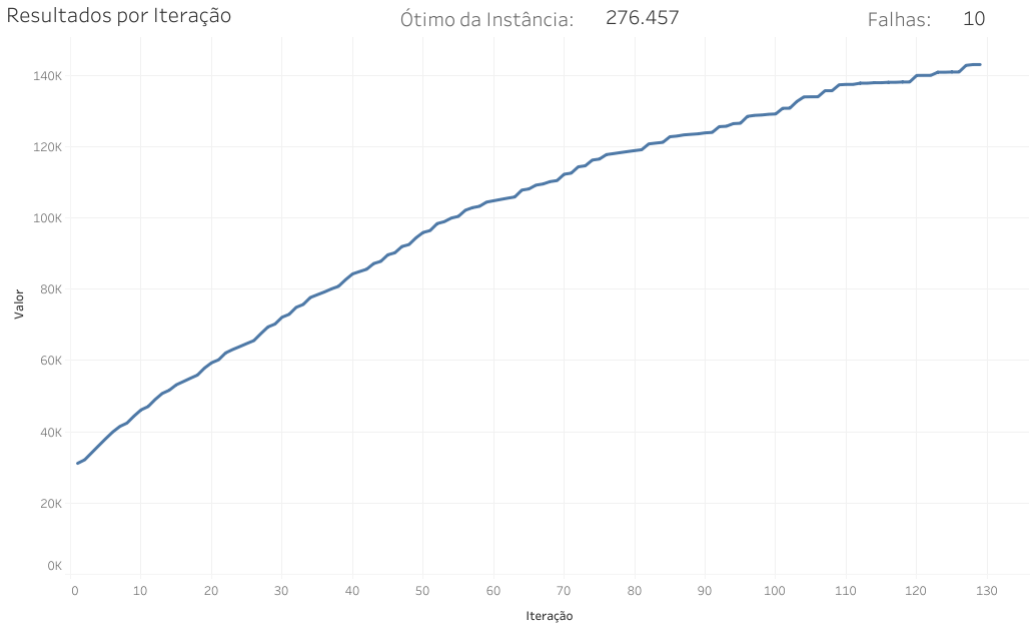

Figura 11: Gráfico - knapPI_1_5000_1000_1

.

Por fim, é possível perceber na prática que a Busca Tabu age em cima das falhas de uma Busca Local. Ao aceitar soluções em que uma melhoria não acontece e em busca de uma melhor solução posterior, a Busca Tabu consegue "fugir"do melhor local e seguir por soluções mais

promissoras, algo que a Busca Local não faz.

O problema da Mochila 0/1 se prova como um contexto interessante para a aplicação de tais algoritmos, visto que esses permitem encontrar soluções cada vez melhores para o problema, sempre respeitando o limite de capacidade. Em bases de dados pequenas, a Busca Tabu mostrou ser fácil encontrar o valor ótimo, porém em bases maiores, por vezes ainda é difícil.

Para encontrar o ótimo em casos onde a base é muito grande o algoritmo ainda depende de muitos fatores, como por exemplo, quanto tempo de execução será possível ser feito, quais são os critérios de parada, como evitar ciclos infinitos e etc. Ainda assim, é uma aplicação possível e promissora para buscar soluções para este problema.

Referências

[Caldas and Ziviani, 2004] Caldas, R. B. and Ziviani, N. (2004). Projeto e análise de algoritmos. *Instituto de Ciências Exatas - UFMG*.

[Ortega, 2022] Ortega, J. (2022). Instances of 0/1 knapsack problem. http://artemisa.unicauca.edu.co/johnyortega/instances_01_KP/. (Acessado em 01/05/2022).

[Talbi, 2012] Talbi, E.-G. (2012). *Metaheuristics from Design to Implementation*. Wiley.