UNIVERSIDADE TUIUTI DO PARANÁ

ANÁLISE DE DESEMPENHO DOS ALGORITMOS TABU SEARCH E SIMULATED ANNEALING APLICADOS NA RESOLUÇÃO DO PROBLEMA DA MOCHILA

CURITIBA

2018

ANDRÉ FELIPE PEREIRA DOS SANTOS

ANÁLISE DE DESEMPENHO DOS ALGORITMOS TABU SEARCH E SIMULATED ANNEALING APLICADOS NA RESOLUÇÃO DO PROBLEMA DA MOCHILA

Trabalho desenvolvido para a disciplina de Inteligência Artificial da Universidade Tuiuti do Paraná.

CURITIBA

2018

SUMÁRIO

1. INTRODUÇÃO

O desenvolvimento e o uso de programas de computadores permitem a realização de tarefas que seres humanos possuem dificuldades em realizar, tais como: o processamento de uma grande quantidade de informações; realizar esse processamento de forma precisa; e resolver problemas complexos. Apesar dos computadores serem eficientes na realização das três tarefas citadas, a complexidade de cada tarefa impacta diretamente no desenvolvimento de programas de computadores, o que exige o uso de técnicas de programação cada vez mais complexas. Este trabalho consiste em realizar uma análise e um comparativo de desempenho entre dois algoritmos de IA (Inteligência Artificial): *Tabu-Search* e *Simulated Annealing*; algoritmos de busca local que são comumente empregados em problemas de otimização, tal como o problema da mochila que será utilizado como base para o desenvolvimento da análise.

2. PROBLEMA DA MOCHILA

O problema da mochila é um problema de otimização, onde se busca otimizar o espaço interno de uma mochila, o qual será preenchido com itens de tamanhos variados. O principal objetivo é preenche-la com o maior número de itens, respeitando o limite máximo de armazenamento da mochila e o tamanho de cada item. Esse problema tem como principal característica, a combinação, pela necessidade de realizar várias tentativas de "encaixe" de cada item, dentro da mochila. Algumas abordagens de resolução desse problema possíveis, porém, nem sempre viáveis, incluem:

- Tentativa e erro (ou enumeração exaustiva): consiste em realizar a combinação de todos os itens, tentando todas as combinações possíveis. Entretanto, essa abordagem não é viável devido ao tempo necessário para se testar todas as combinações, pois conforme o número de itens aumenta, o número de combinações também aumenta de forma muito elevada, porém de forma muito elevada, podendo levar séculos processar todas as combinações.

- *Backtracking*: consiste em seguir a mesma abordagem do primeiro, porém gerando as combinações de forma progressiva e conforme necessário, visando excluir combinações que não geram um resultado válido. As combinações são geralmente estruturadas no formato de árvore e utiliza-se a busca em profundidade ou busca local para realizar a varredura nesta árvore por alguma solução válida e nem sempre ótima. Os dois algoritmos citados (*Tabu-Search* e *Simulated Annealing*), pertencem a esta abordagem.

- Aproximação de *Greed*: consiste em ordenar os itens de forma crescente de tamanho e escolhê-los até que a mochila esteja cheia. Por ser uma abordagem gulosa, esta abordagem pode resultar em uma solução sub-ótima, assim como a abordagem anterior.

- Programação Dinâmica: sua abordagem é semelhante ao da primeira abordagem, tentativa e erro, entretanto com o armazenamento de soluções anteriores que são consultadas a todo o tempo, evitando a geração de combinações repetidas.

3. ALGORITMOS DE BUSCA LOCAL

Os algoritmos de busca local buscam encontrar uma solução possível, mas nem sempre ótima, para um problema definido. Têm como principal característica a varredura de um espaço de busca, como um grafo ou uma árvore, a partir de um ponto inicial escolhido de forma aleatória ou através de uma heurística, informações conhecidas a respeito do problema a ser resolvido. Dois desses algoritmos são o *Tabu-Search* e o *Simulated Annealing*.

3.1. TABU-SEARCH

> "*Tabu-search* é uma meta-heurística que orienta um procedimento de busca heurística local para explorar o espaço da solução além da otimização local. Um dos principais componentes do *Tabu Search* é o uso de memória adaptativa, que cria um comportamento de pesquisa mais flexível" (FRED GLOVER, 1988).

O algoritmo *Tabu-Search* utiliza o conceito de memória adaptativa para armazenar a solução encontrada anteriormente para determinar qual "caminho", ou solução seguinte irá analisar, que deverá ser melhor que a atual. O algoritmo *Tabu-search* é descrito na Figura 1.

3.2. SIMULATED ANNEALING

O algoritmo *Simulated Annealing* surgiu como uma alternativa ao *Tabu-search*, por sua abordagem sempre escolher as melhores soluções, desde o início do algoritmo. O *Simulated Annealing* permite uma tolerância maior com as soluções ruins, encontradas no início do algoritmo, permitindo um alcance maior de busca.

> "*Simulated Annealing* é uma forma eficaz e geral de otimização. É útil para encontrar ótimos globais na presença de um grande número de ótimos locais. "*Annealing*" refere-se a uma analogia com a termodinâmica, especificamente com a maneira como os metais esfriam e recozem. O *Simulated Annealing* usa a função objetiva de um problema de otimização em vez da energia de um material." (BOYAN, 1995, adaptado).

O algoritmo *Simulated Annealing* consiste em dois passos:

1. Incrementa a temperatura do sistema até um valor limite;
2. Decrementa a temperatura do sistema até um valor limite.

Em ambos os passos, é feita a seleção da solução na vizinhança da solução atual, existente no espaço de busca, utilizando a Equação *1*. O algoritmo *Simulated Annealing* pode ser visto na Figura 2.

Equação 1 - Métrica de desempenho

$$Métrica\ Solução = \mathrm{e}^{\left(\frac{S_{atual} - S_{vizinho}}{Temperatura}\right)}$$

Fonte: (JAVAD SALIMI, 2016, p. 4).

Figura 1- Algoritmo *Tabu Search*

```
def tabu_search(espaco_busca):
    # escolher uma solução inicial (que será a atual), existente no espaço de busca, dada uma regra definida.
    solucao_atual = solucao_inicial(espaco_busca)

    # define uma lista tabu vazia.
    lista_tabu = []

    # enquanto não atingo a condição de parada.
    while not condicao_parada(solucao_atual):

        # seleciona os vizinhos melhores que a solucao_atual.
        for vizinho in solucao_atual.vizinhanca:
            if vizinho.valor > solucao_atual.valor:
                lista_tabu.append(vizinho)

        # ordena pelo valor da solução e escolhe a melhor solução das encontradas.
        lista_tabu.sort(lambda x: x.valor)
        solucao_atual = lista_tabu[0]

    return solucao_atual
```

Fonte: (TANZILA ISLAM, 2016, p. 5)

Figura 2 - Algoritmo *Simulated Annealing.*

```
def simulated_annealing(temperatura_inicial):
    # escolher uma solução inicial e uma global(que será a atual), existentes no espaço de busca, dada uma regra definida.
    solucao_atual  = solucao_inicial()
    melhor_solucao = solucao_atual

    # defino a temperatura inicial
    temperatura_atual = temperatura_inicial

    # enquanto não atingo a condição de parada.
    while not condicao_parada(solucao_atual):

        # seleciona os vizinhos melhores que a solucao_atual.
        for vizinho in solucao_atual.vizinhanca():

            # se o vizinho for melhor ou igual à solução atual, faço ele a solução atual.
            if vizinho >= solucao_atual:
                solucao_atual = vizinho

                # se o vizinho for melhor que a melhor solução, faço ele a melhor solução
                if vizinho >= melhor_solucao:
                    melhor_solucao = vizinho

            # se o resultado da equação for maior que o valor sorteado, faço ele a solução atual.
            # quanto maior a temperatura, maior a chance dele escolher uma solução ruim como solução atual.
            # isso serve para realizar a busca em seus vizinhos, para "pular" os mínimos/máximos locais,
            # visando atingir o mínimo/máximo global.
            elif exp((solucao_atual - vizinho) / temperatura_atual) > random():
                solucao_atual = vizinho

        # calcula a temperatura
        atualiza_temperatura(temperatura_atual)
    return melhor_solucao
```

Fonte: (JAVAD SALIMI, 2016, p. 4)

4. RESULTADOS EXPERIMENTAIS

4.1. CENÁRIOS DE TESTE

Para ambos os algoritmos foram utilizados os seguintes parâmetros:

- Capacidade limite da mochila: 100 e 1000;
- Número de itens: 100 e 1000.

Os critérios de avaliação utilizados foram:

- Tempo de execução (melhor e pior);
- Capacidade média obtida;
- Média de número de itens obtida.

Para a geração dos resultados, cada algoritmo foi executado 5 vezes, ambos com o mesmo estado inicial. Também foi utilizado em ambos os algoritmos, como critério de parada, a estratégia de número de iterações sem melhora. Para o *simulated annealing*, foi utilizado como temperatura o valor 5000, com decaimento de 1.

4.2. RESULTADOS

Os resultados obtidos podem ser visualizados nos Quadros de 1 a 4:

Quadro 1 - Resultados para o número de itens=100 e capacidade limite=100

Parâmetros	**Tabu Search**	**Simulated Annealing**
Número de Itens	100	100
Capacidade Limite Mochila	100	100
Capacidade Média Obtida	100	97
Média de Número de Itens Obtida	9,6	2
Melhor Tempo Execução	0,125	6,0625
Pior Tempo Execução	0,1562	6,4218

Fonte: Próprio autor, 2018

Quadro 2 - Resultados para o número de itens=1000 e capacidade limite=100

Parâmetros	**Tabu Search**	**Simulated Annealing**
Número de Itens	1000	100
Capacidade Limite Mochila	1000	100
Capacidade Média Obtida	100	75,2
Média de Número de Itens Obtida	25,2	2
Melhor Tempo Execução	3,1562	139,2
Pior Tempo Execução	4,1406	146,1

Fonte: Próprio autor, 2018

Quadro 3 - Resultados para o número de itens=100 e capacidade limite=1000

Parâmetros	**Tabu Search**	**Simulated Annealing**
Número de Itens	100	1000
Capacidade Limite Mochila	100	1000
Capacidade Média Obtida	996	158,4
Média de Número de Itens Obtida	43,4	2
Melhor Tempo Execução	0,2343	6,2656
Pior Tempo Execução	0,25	6,7031

Fonte: Próprio autor, 2018

Quadro 4 - Resultados para o número de itens=1000 e capacidade limite=1000

Parâmetros	**Tabu Search**	**Simulated Annealing**
Número de Itens	1000	1000
Capacidade Limite Mochila	1000	1000
Capacidade Média Obtida	602,8	101
Média de Número de Itens Obtida	101	2
Melhor Tempo Execução	6,5937	141
Pior Tempo Execução	7,2968	148,7

Fonte: Próprio autor, 2018

4.3. ANÁLISE DOS RESULTADOS

Analisando os quadros de 1 a 4, existentes nas figuras 1 e 2, percebe-se que o algoritmo *tabu search* teve um desempenho melhor em todos os quesitos analisados. Percebe-se também que o aumento do número de itens impacta mais o desempenho dos dois algoritmos testados do que o tamanho da mochila.

5. CONCLUSÃO

Analisando a implementação dos dois algoritmos analisados, *tabu search* e *simulated annealing*, observa-se que o primeiro possui uma estrutura mais simples que o segundo, em termos de estrutura de controle, que no caso do segundo, utiliza-se a temperatura inicial e o decaimento da mesma como controle. Observa-se a partir dos dados existentes nos quadros de 1 a 4, o fato do algoritmo *simulated annealing* ter um tempo de execução muito maior que o *tabu search*. Este fenômeno pode ser explicado pela estrutura do algoritmo em si, que seleciona a pior solução com maior frequência, quando a temperatura é mais alta, fazendo com que o algoritmo execute por mais tempo e aumentando o risco da solução ser ruim, caso a temperatura inicial seja muito alta e o decaimento muito pequeno.

O último fenômeno percebido é o grande impacto que a seleção da solução inicial tem sobre o resultado final, em ambos os algoritmos. Devido a isso, sugere-se o uso de meta-heurísticas para se determinar uma boa solução inicial, visando obter uma melhora na melhor solução encontrada.

REFERÊNCIAS

BOYAN, J. A. What is Simulated Annealing, 1995. Disponivel em: <http://www.cs.cmu.edu/afs/cs.cmu.edu/project/learn-43/lib/idauction2/.g/web/glossary/anneal.html>. Acesso em: 12 novembro 2018.

FRED GLOVER, R. M. **Tabu Search**. University of Colorado. Boulder, p. 17. 1988.

TANZILA ISLAM, Z. S. M. A. P. M. H. University Timetable Generator Using Tabu. **Journal of Computer and Communications**, Dhaka, 26 dezembro 2016. 28-37.