Pontifícia Universidade Católica
do Rio de Janeiro

Inteligência Artificial

Bin Packing Problem

Solucionando o problema através de um algoritmo de Busca Heurística e de um algoritmo com Busca Local.

Bruce Marcellino

Bruno Miranda Marinho

Professora Renatha Capua

INF1771 – Inteligência Artificial – 2017.2 – 3WA

1. Introdução

Este é um problema considerado NP-Difícil, na teoria de complexidade computacional, pelo fato de não ser conhecido um algoritmo que resolva o problema em tempo polinomial.

Deste problema, surgiram muitas variações, como por exemplo o empacotamento linear (LP), empacotamento por custo (CP), empacotamento por peso (WP). As aplicações destes problemas são muitas, desde o preenchimento de recipientes até carregamento de caminhões com capacidade e/ou peso restrito.

2. Definição do Problema

O Bin Packing Problem consiste em alocar itens de diferentes pesos em caixas com capacidade limitada, de forma a minimizar a quantidade de caixas utilizadas para acomodar todos os itens. A alocação dos itens deve respeitar a sua capacidade, e na solução proposta, nenhuma caixa pode conter mais do que ela suporta.

Nesta instância do problema, não são considerados conflitos (onde certos itens não podem ser alocados juntos com outros determinados itens, exceto se isso implicar em ultrapassar a capacidade da caixa). A única restrição a ser levada em consideração é a capacidade das caixas (todas possuem a mesma capacidade).

3. Metodologia

Para garantir o funcionamento completo do programa, incluindo a leitura do arquivo de entrada, que contém a quantidade de itens, a capacidade das caixas e em seguida, os pesos dos itens (que são numerados de acordo com sua ordem), é necessário que o arquivo de entrada siga o seguinte formato:

```
70      <- Quantidade de itens
1000    <- Capacidade das caixas
765     <- Pesos dos itens, a seguir.
567
...
230
```

Para executar o programa, é necessário executar através do prompt de comando (Windows) ou através do terminal (Linux), pois o programa toma como argumento o caminho do arquivo de entrada, e opcionalmente a semente.

Veja o passo a passo (com algumas sugestões) abaixo, para executar os programas sem problemas.

Algoritmo I (Dispersar)

Passo a Passo da Execução

WINDOWS

Crie uma pasta chamada “Dispersar” e nela coloque o executável Dispersar.exe.

Se preferir, crie uma pasta chamada “instancias”, para colocar as instancias que serão lidas pelo programa.

Para executar o programa, navegue até a pasta “Dispersar” no prompt de comando e digite o nome do executável e o caminho do arquivo a ser lido, como por exemplo:

Dispersar instancias/nome_da_instancia

Será criado um arquivo “resultado.sol” na pasta “resultado”.

O tempo de execução será exibido no prompt de comando.

LINUX

Crie uma pasta chamada “Dispersar” e nela coloque o executável Dispersar.

Se preferir, crie uma pasta chamada “instancias”, para colocar as instâncias que serão lidas pelo programa.

Para executar o programa, navegue até a pasta “Dispersar” no terminal e digite o nome do executável e o caminho do arquivo a ser lido, como por exemplo:

./Dispersar instancias/nome_da_instancia

Será criado um arquivo “resultado.sol” na pasta “resultado”.

O tempo de execução será exibido no terminal.

Detalhamento do Algoritmo

Este algoritmo percorre os itens, tentando colocar cada item na primeira caixa disponível. Se nenhuma caixa tiver capacidade para abrigar o item, ele é colocado numa caixa vazia. É um algoritmo guloso porque segue o primeiro caminho que seja aparentemente o melhor para chegar à solução.

```
1 Para cada item
2     Para cada caixa
3         Se este item cabe nesta caixa
4             Coloco este item nesta caixa
5             CapRestante <- CapRestante - PesoItem
6     Se este item não coube em nenhuma caixa existente
7         Crio uma nova caixa
8         Coloco este item nesta caixa
9         CapRestante <- CapRestante - PesoItem
```

Onde “CapRestante” é a Capacidade Restante da caixa corrente e “PesoItem” é o Peso do Item corrente.

Algoritmo II (Complementar)

Passo a Passo da Execução

WINDOWS

Crie uma pasta chamada “Complementar” e nela coloque o executável Complementar.exe.

Se preferir, crie uma pasta chamada “instancias”, para colocar as instancias que serão lidas pelo programa.

Para executar o programa, navegue até a pasta “Complementar” no prompt de comando e digite o nome do executável e o caminho do arquivo a ser lido, como por exemplo:

Complementar instancias/nome_da_instancia

Este algoritmo pode ser executado com outros argumentos também, por exemplo:

Complementar path/nome_da_instancia semente
ou
Complementar path/nome_da_instancia semente max

Onde 'semente' será usada na função random e 'max' irá controlar o número máximo de iterações até que uma solução não-ótima seja aceita.

Será criado um arquivo “nome_da_instancia.sol” na pasta “resultado”.

O tempo de execução será exibido no prompt de comando.

LINUX

Crie uma pasta chamada “Complementar” e nela coloque o executável Complementar.

Se preferir, crie uma pasta chamada “instancias”, para colocar as instâncias que serão lidas pelo programa.

Para executar o programa, navegue até a pasta “Complementar” no terminal e digite o nome do executável e o caminho do arquivo a ser lido, como por exemplo:

./Complementar instancias/nome_da_instancia

Este algoritmo pode ser executado com outros argumentos também, por exemplo:

./Complementar path/nome_da_instancia semente
ou
./Complementar path/nome_da_instancia semente max

Onde 'semente' será usada na função random e 'max' irá controlar o número máximo de iterações até que uma solução não-ótima seja aceita.

Será criado um arquivo “nome_da_instancia.sol” na pasta “resultado”.

O tempo de execução será exibido no terminal.

Detalhamento do Algoritmo

Este algoritmo combina uma busca heurística para encontrar uma solução inicial com uma busca local para procurar soluções vizinhas.

Busca Heurística:

Os itens são divididos em dois grupos, no primeiro grupo ficam os itens de maior peso, e no segundo os de menor peso. Tenta-se combinar cada item de um vetor com os itens do outro vetor.

```
1 Ordena o vetor de itens
2 Para cada item
3       Se este estiver na primeira metade do vetor
4             Este é colocado no vetor A
5       Se este estiver na segunda metade
6             Este é colocado no vetor B
```

Assim é feita a divisão.

```
1 Para cada item do vetor A
2       Ele é colocado numa caixa A
3       Para cada item do vetor B
4             Se este cabe na caixa A
5                   Este é colocado na caixa A
```

Até que todos os itens de A tenham uma caixa

Podem sobrar itens no vetor B que não possuem caixas, esse serão colocados em uma caixa de forma semelhante ao algoritmo de dispersar.

E assim é encontrada uma solução inicial.

Busca local:

Na busca local, duas operações são realizadas para gerar os vizinhos, troca de itens, e troca de caixa. Trocar um item é fazer um *swap* entre itens de caixas diferentes. Trocar de caixa quer dizer realocar um item de uma caixa para outra, sem retirar um item daquela.

Um item é selecionado aleatoriamente, e tenta-se trocá-lo com cada um dos itens em uma caixa diferente dele. Assim são gerados os vizinhos, o vizinho que deixar uma caixa mais cheia possível é o eleito como próximo estado.

Existe, entretanto, uma outra forma de gerar vizinhos, que é trocando o item selecionado de uma caixa para outra, na qual ele caiba, e que esteja mais cheia do que a que ele está. Essa forma recebe prioridade sobre a anterior, pois é através dela que é possível esvaziar caixas.

Um parâmetro é usado para determinar quantas trocas de estados serão feitas até que uma solução seja considerada aceitável.

A imagem abaixo, mostra o pseudo-código da busca local.

```
1 Um item A é selecionado aleatoriamente
2       Para cada caixa
3             Se colocar o item A naquela caixa resulta num estado melhor do que o atual
4                   Tal estado é guardado como estado A
5       Se estado A não foi definido
6             Para cada item B em um caixa diferente
7                   Se trocar item A com item B resulta num estado melhor que o atual
8                         Se esse estado é melhor do que o estado A
9                               Tal estado é guardado como estado A
10 O estado é mudado para o estado A.
```

Arquivo de Saída

Para este algoritmo, o arquivo de saída "nome_da_instancia.sol" será gerado numa pasta chamada "resultado", que estará no mesmo local que o executável, e segue o seguinte formato:

Quantidade de caixas utilizadas (Primeira linha do arquivo)

Conteúdo de cada caixa (Linhas seguintes)

4. Resultados

Para o algoritmo "**Dispersar**", os resultados obtidos foram em média para cada instância:

Formato:

Instância

Número de caixas – Tempo médio de execução

Falkenauer – 60 itens	
23 caixas	0.003 segundos
Falkenauer – 120 itens	
45 caixas	0.003 segundos
Falkenauer – 120 itens (2)	
47 caixas	0.002 segundos
Falkenauer – 250 itens	
102 caixas	0.002 segundos
Falkenauer – 500 itens	
207 caixas	0.003 segundos

Para o algoritmo “**Complementar**”, os resultados obtidos foram em média para cada instância:

Formato:

Instância

Número de caixas – Tempo médio de execução – Semente gerada - Max

Falkenauer – 60 itens			
21 caixas	0.003 segundos	1506889563	1000 (Default)
Falkenauer – 120 itens			
41 caixas	0.004 segundos	1506889654	1000(Default)
Falkenauer – 120 itens (2)			
47 caixas	0.004 segundos	1506889744	1000(Default)
Falkenauer – 250 itens			
102 caixas	0.004 segundos	1506889832	1000(Default)
Falkenauer – 500 itens			
208 caixas	0.006 segundos	1506889886	1000(Default)

5. Conclusão

Foram apresentados dois algoritmos para solucionar o problema do Bin Packing, sendo o primeiro deles, uma busca gulosa, baseado no conceito de alocar um item na primeira caixa possível. O segundo algoritmo, encontrou uma solução inicial, procurando os encaixes perfeitos ou mais próximos disso e então procurou melhorá-la através de uma busca local, gerando vizinhos e os analisando. Este último, apresentou resultados satisfatórios para todas as instâncias.

O primeiro algoritmo, apresentou resultados bem piores para as instâncias menores, e resultados aproximados nas instâncias maiores, comparado ao segundo algoritmo, que só se mostrou pior na última instância de teste (Falkenauer_u500_05), com a diferença de uma caixa.

Durante o processo de confecção deste trabalho, pudemos aprender na prática o funcionamento de dois algoritmos básicos de Inteligência Artificial. Praticamos formas de estruturar dados para que o computador pudesse interpretá-los. Exploramos a escolha entre obter rapidamente um resultado imperfeito e nos permitir mais tempo para um resultado melhor. E aprendemos que uma solução é apenas o ponto de partida de um novo problema.