Implementação de heurísticas em *Prolog* para o problema do contâiner

Bruno Barros Mello[1]
Juliana Moura[1]
Vitor Balestro[1,2]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brasil

[2]Instituto de Matemática e Estatística – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brasil

brunobarrosmello@id.uff.br, juliana_moura@id.uff.br, vitorbalestro@id.uff.br

Abstract. *O problema do contâiner busca a melhor configuração de um conjunto de caixas em um contâiner. Este trabalho é dedicado a implementar soluções heurísticas em* Prolog *para este problema.*

1. Introdução

Neste trabalho, discutimos algumas implementações de heurísticas na linguagem *Prolog* para resolver o *problema do contâiner*. O enunciado do problema é o seguinte: dados um contâiner e um conjunto de caixas com dimensões especificadas (C x L x A), determinar a configuração de caixas no contâiner que maximize a quantidade de caixas empacotadas ou que minimize o volume restante no contâiner. Em nossas abordagens, adotamos o primeiro critério.

Nossa primeira estratégia é um algoritmo guloso simples que não envolve ordenamento das caixas e usa uma regra simples para estabelecer o espaço disponível no contâiner. Entretanto, esta regra é excessivamente simples, e não considera realmente a geometria tri-dimensional do problema. Esta heurística deve ser entendida com fins didáticos.

A segunda estratégia é um algoritmo de *First Fit Decreasing*, sem rotação. Esta heurística ordena as caixas em ordem decrescente de volume e percorre a lista de caixas verificando todas as posições possíveis em que a caixa atual pode ser encaixada no contâiner (a partir de verificações de sobreposição com as outras caixas já empacotadas). Ao final, a melhor solução é selecionada (o critério usado é o número de caixas empacotadas). Esta heurística é implementada com ou sem considerar possíveis rotações das caixas.

Além destas duas estratégias, também implementamos um algoritmo de *Largest Area First Fit*. Nesta heurística, as caixas são ordenadas pela maior área da base. Para cada caixa, o algoritmo identifica as posições de encaixe disponíveis e escolhe aquela com a menor altura. Caixas que não podem ser encaixadas são ignoradas.

Todos os códigos implementados podem ser encontrados no repositório do trabalho: https://github.com/vitorbalestro/TrabalhoProlog. Uma boa referência para o problema do contâiner é [**?**].

2. Algoritmo guloso simples

Neste algoritmo, as caixas devem ser consideradas em uma lista não-ordenada. As caixas são construídas como predicados da forma

```
caixa(ID, C, L, A),
```

e a lista é obtida através do método *findall*. O contâiner é declarado pelo predicado

```
container(C, L, A).
```

O estado inicial do problema é a declaração do contâiner e das caixas. O algoritmo constrói a lista de caixas e percorre esta lista verificando se cada caixa pode ser incluída no contâiner. A regra para isto é simples: o algoritmo guarda a soma das dimensões de todas as caixas anteriormente adicionadas, e compara as dimensões da caixa atual com os valores "restantes"no contâiner. Se a caixa atual não puder ser inserida segundo este critério, esta caixa é ignorada e a próxima caixa da lista é lida.

Note que esta estratégia pode ser muito ineficiente. Se uma das dimensões for "completada", então o algoritmo não aceitará nenhuma caixa adicional, ainda que sobre muito espaço no contâiner. Além disso, esta implementação não considera rotações das caixas.

O estado final do algoritmo é a lista das caixas que foram incluídas no contâiner. No repositório do trabalho, este algoritmo está no arquivo greedy.pl.

3. *First Fit Decreasing*

Neste algoritmo, cada caixa é declarada como um predicado do tipo

```
box(W, H, D),
```

e o contâiner é declarado pelo predicado

```
container(W, H, D).
```

As caixas declaradas em uma lista, pelo predicado

```
boxes([box(W1, H1, D1), box(W2, H2, D2), ...]
```

que, posteriormente, é ordenada por volumes decrescentes. O objetivo é dar prioridade às caixas maiores, que são mais difíceis de empacotar nos espaços restantes.

A lista ordenada de caixas é percorrida, e a cada etapa o algoritmo tenta encontrar uma posição para a caixa atual usando o método *between*. Cada posição potencial é testada para sobreposição com as caixas que já estão alocadas no contâiner.

Todas as soluções possíveis são armazenadas e, ao final, o algoritmo elege a melhor solução. O critério para isto é a quantidade de caixas empacotadas. Desta forma, o estado final é uma lista de caixas junto com suas posições de empacotamento (isto é, as coordenadas dos respectivos cantos inferiores).

Esta heurística também não considera as possíveis rotações das caixas. Isto é, cada caixa só pode ser empacotada em sua posição declarada. No respositório do trabalho, este algoritmo encontra-se no arquivo ffd.pl.

4. FFD com rotações

Este algoritmo é idêntico ao anterior, exceto por ele considerar rotações das caixas. Assim, o algoritmo verifica as posições em que cada rotação de cada caixa pode ser empacotada. O código desta implementação encontra-se no arquivo ffd_with_rotation.pl. No arquivo ffd_with_rotation_2.pl, encontra-se o mesmo código mas com um estado inicial mais complexo.

5. Extra: visualização da solução com Minecraft

Uma dificuldade que o grupo teve com este trabalho foi conseguir visualizar facilmente a solução gerada pelos algoritmos, para validar que está correta. Para isso, uma vez que a solução foi gerada adicionamos uma seção de impressão de comandos *fill* do jogo Minecraft para conseguir preencher uma região com blocos de diferentes cores de acordo com a solução gerada pelo algoritmo. Uma demonstração desta visualização pode ser encontrada aqui.

Esta visualização foi utilizada em todos os métodos, mas para não poluir os arquivos de solução com código repetido de coisas extras, manteremos o código desta parte apenas no arquivo *ffd_with_rotation.pl*. Note que o sistema de coordenadas do jogo utiliza o eixo Z invertido, então coordenadas (X, Y, Z) viram $(X, Y, -Z)$ no jogo.

6. *Largest Area First Fit*

O estado inicial do problema é o mesmo da heurística FFD: um contâiner tri-dimensional e uma coleção de caixas tri-dimensionais. Primeiro, as caixas são ordenadas por área da base decrescente. Então, o algoritmo percorre esta lista ordenada tentando empacotar cada caixa. Para isto, o algoritmo testa todas as posições possíveis de empacotamento, e seleciona aquela que tem a menor altura. Esta heurística foi inspirada em [**?**].

7. Resultados e Discussão

Executamos o método guloso e o método LAFF para um contâiner de 50 x 50 x 50 e 30 caixas. Conforme esperado, o método guloso teve uma performance muito ruim: apenas 5 caixas foram empacotadas. O método LAFF, por outro lado, conseguiu empacotar todas as caixas (solução ótima, sob nosso ponto de vista de quantidade de caixas empacotadas). Ao reduzir o contâiner para 20 x 20 x 20, o LAFF empacotou 13 caixas, e o algoritmo guloso empacotou apenas 3. O tempo de execução para o método guloso é desprezível. O tempo de execução do método LAFF nos dois testes ficou na ordem de segundos.

O método FFD apresentou menor eficiência do que o método LAFF. A tentativa de execução deste método para 30 caixas resultou em estouro de pilha. Para um estado inicial de 15 caixas em um contâiner de 20 x 20 x 20, o método FFD com rotações empacotou 13 caixas (assim como o LAFF). Entretanto, o tempo de execução está na ordem dos minutos.

A razão para a menor eficiência do FFD possivelmente estã no fato de que esta heurística considera **todas** as posições possíveis de empacotamento e, portanto, cria (abstratamente) uma árvore de possíveis empacotamentos cuja quantidade de nós é exponencial no tamanho do container. Depois, o algoritmo seleciona a melhor delas. O método LAFF, por outro lado, considera uma escolha gulosa: dadas as posições possíveis (de cada

etapa), apenas uma delas é selecionada segundo um critério (especificamente, a menor altura da posição).

Em resumo, os testes executados indicam que a heurística LAFF é mais eficiente e não tem perda significativa de qualidade em relação à heurística FFD.

Referências

[1] M. Gürbüz, S. Akyokus, I. Emiroglu, A. Güran: An Efficient Algorithm for 3D Rectangular Box Packing. *Applied Automatic Systems Proceedings of Selected AAS*. Ohrid, 26–29.09.2009. Published by 'Society of ETAI of Republic of Macedonia, Skopje, 2009.

[2] B. K. A. Ngoi, M. L. Tay, E. S. Chua: Applying spatial representation techniques to the container packing problem. *Int. J. Prod. Res.* **32** (1), pp. 111–123, 1994.