Procura e Planeamento

Alameda Campus
Projecto (2015/2016)

Grupo: 18

Nome: Daniel Brandão Número:76245
Nome: Miguel Duarte Número: 85016

Nota:

Soma das horas investidas somente no desenvolvimento deste trabalho: 40h

Entrega em: 23h59, 4 de Dezembro 2015.

Índice

1. Introdução

1.1. Descrição do Modelo

Decidimos representar cada estado do problema como uma estrutura *rect* que contém: *pecas-i*, que são as peças que ainda faltam colocar (no inicio do problema esta inclui as peças todas, no estado objectivo este conjunto é vazio); *pecas-f*, são as peças colocadas com as respectivas posições e orientações (no inicio do problema este é o conjunto vazio e no estado objectivo é o conjunto de todas as peças); *width*, denota o comprimento do rectângulo; *height*, denota a largura do rectângulo; *posicoes*, conjunto de posições onde iremos colocar uma peça. Inicialmente *posicoes* é inicializada com a posição (0, 0). Quando uma peça é colocada, a posição onde foi colocada e retirada do conjunto de posições e são adicionadas pelo menos duas novas posições que correspondem às posições imediatamente em cima e à direita da peça colocada; por exemplo, colocamos a peça de comprimento 3, largura 2, orientação H na posição (0,0), então iremos adicionar ao conjunto *posicoes* as posicões (3;0) e (0;2) se ainda estiverem nos limites do rectângulo, retirando (0;0) deste conjunto.

Para problemas de optimização, colocamos a *height* como *most-positive-fixnum*, para simular uma altura infinita.

Para problemas de satisfação, no estado inicial, verificamos se a área total das peças excede a área do rectângulo e se existe alguma peça com dimensões que excedem as do rectângulo. Se estas condições se verificarem, o conjunto *posicoes* é inicializado a vazio e isto fará o *operator*(função que gera os próximos estados) não encontrar solução para o problema, logo na primeira iteração. Esta optimização permite aos algoritmos de satisfação darem resposta instantânea para problemas que são obviamente sem solução.

A função *operator* gera estados descendentes de acordo com as peças que ainda faltam encaixar, as diferentes posições do conjunto *posicoes* e as orientações horizontal e vertical, retornando o conjunto de estados onde é possível atribuir uma peça a uma posição com uma determinada rotação. Esta averiguação é feita na função *encaixa-peça-?*, que verifica se as peças poderão ficar fora do rectângulo ou se colidem com outras peças já encaixadas. O novo estado gerado é obtido na função *encaixa,* que faz uma cópia dos atributos do estado anterior e modifica-os de acordo com o que foi descrito no primeiro parágrafo, mantendo o comprimento e altura do rectângulo.

1.2. Descrição das estruturas de dados

As estruturas de dados usadas para representar as *pecas-i*, *pecas-f* e *posicoes* foram as listas, visto que usualmente quando fazemos um acesso a estas, percorremos o conjunto inteiro, não sendo necessário procurar por elementos específicos dentro destes conjuntos. Desta forma, decidiu-se que listas seria a estrutura de dados mais adequada para representar estes conjuntos, não se utilizando *arrays* nem *hashtables*.

1.3. Descrição das heurísticas

a. Heurística baseada na Área

Esta heurística chamada *h-area,* baseia-se em atribuir a cada estado a diferença entre o valor da área total do rectângulo e a soma das áreas das peças já atribuídas. Após alguns testes, chegamos à conclusão que esta heurística funciona bem para problemas com algum espaço livre mas demora a dar solução para problemas onde a solução é apertada, isto é, não existirá qualquer espaço entre as peças colocadas, o conjunto de todas as peças deve corresponder à área do rectângulo. O problema desta heurística para problemas onde a área das peças é muito parecida e o facto dos estados com diferentes posições e orientações terem o mesmo valor (visto que só estamos a ter em conta a área das peças colocadas), não criará muita distinção entre estados à mesma profundidade.

b. Heurística baseada no Comprimento

Esta heurística chamada *h-comp*, atribui a cada estado o simétrico da soma dos maiores comprimentos das peças já atribuídas, priorizando ao início as peças com maior comprimento (independentemente da área). Esta heurística apresentou os mesmos problemas que a heurística acima para problemas com solução apertada e onde os comprimentos das peças não varia muito.

c. Heurística Optimista

Esta heurística de nome *complicated,* para cada estado, percorre as peças por atribuir e o conjunto de possíveis posições de atribuição. Se existir uma peça cuja sua dimensão mínima não for menor ou igual à distância entre a posição e cada uma das extremidades do rectângulo, então digo que o valor deste estado é infinito (*most-positive-fixnum)* e não quero percorrê-lo; se não, incremento um contador que será no final, se a condição se verificar para todas as peças e posições, a multiplicação do número de peças atribuir e o número de posições onde as posso atribuir. Esta heurística é muito optimista e afunda rapidamente e encontra soluções rapidamente para problemas apertados. Irá cortar estados que poderiam levar a uma solução, sacrificando estes caminhos para afundar e chegar à solução. Problema desta heurística e que não funciona bem para problemas não apertados visto que, havendo mais espaço, menos estados serão cortados, e como o valor do contador será igual para estados do mesmo nível, torna-se extremamente ineficiente. Combinada com as heurísticas acima perde a capacidade de resolver os problemas apertados rapidamente ou torna-se muito especifica para um problema resolvendo mal os restantes.

d. Heurística baseada na Altura

Esta heurística de nome *h-height,* atribui a cada estado o valor heurístico correspondente à sua altura. A altura é calculada percorrendo todas as peças já atribuídas e

calcular a altura que atingem, sendo a altura retornada a maior destas. Esta heurística é usada para problemas de optimização, onde o objectivo e minimizar esta altura.

e. Heurística baseada nos espaços em branco

Esta heurística, de nome *h-hole*, calcula o número de espaços não preenchidos, rodeados por peças por todos os lados. Quanto mais espaços houver, maior será o valor retornado e portanto pior será o estado.

2. Problema de Satisfação

2.1. Procuras do *procura.lisp*

Das procuras do ficheiro procura.lisp, consideramos usar para estudar a procura em profundidade e A*. Não consideramos a procura com profundidade iterativa visto que as soluções encontram-se todas à mesma profundida e, pelas mesmas razões, não consideramos o IDA* nem a procura em largura, visto que o objectivo é afundar e chegar rapidamente aos estados solução e não perder tempo a procurar estados nos primeiros nós da árvore. Como a procura em profundidade é uma procura *brute force*/não-informada, este e apenas escolhido apenas para comparação com a procura informada A* e as outras procuras implementadas.

2.2. Procura Sondagem Iterativa

A sondagem iterativa é implementada com a função principal *i-sampling-sat* e com a função secundária *samp.* Esta última implementa a estratégia 1-samp, que faz uma sondagem aleatória no espaço de procura, até chegar a um nó objectivo ou encontrar um nó sem descendentes. A função principal chama esta até que *samp* retorne uma solução

2.3. Procura ILDS

A procura ILDS para o problema de satisfação é implementada com recurso às funções *ILDS* e *LDS*. A função ILDS chama a função LDS (Limited Discrepancy Search) para executar todas as pesquisas com exatamente n discrepâncias. Esta propriedade é mantida através da verificação do número de discrepâncias relativamente à heurística e a profundidade que ainda falta para chegar às folhas.

2.4. Comparação

Os algoritmos corridos foram limitados a um período de 5 minutos.

Visto que o *operator* é o mesmo e que as soluções estão todas ao mesmo nível da árvore, os valores de profundidade máxima, profundidade da solução, fator de ramificação serão exatamente ou aproximadamente da mesma ordem de grandeza para todas as procuras.

Decidimo-nos, desta forma, focarmo-nos no tempo de procura que cada procura demora a retornar uma solução.

Os casos usados para testar foram criados aleatoriamente com 10, 20, 30, 35, 40 e 50 peças com dimensões compreendidas entre 1 e 5. Usamos inicialmente os algoritmos de optimização para ajustar as dimensões do rectângulo de forma a obter problemas não triviais.

	10 peças	**20 peças**	**30 peças**	**35 peças**	**40 peças**	**50 peças**
Profundidade	t=0.014s	t=0.04s	t=0.1572s	t > 5min.	t > 5min.	t > 5min.
A* h-comp	t=0.028s	t=0.09s	t=0.403s	t=0.640s	t=0.836s	t=2.19s
A* h-area	t=0.03s	t=0.076s	t=0.390s	t=0.577s	t=0.799s	t=1.967s
ILDS h-area	t=0.013s	t=0.041s	t=0.233s	t=2.703s	t=7.937s	t=53.656s
ILDS h-comp	t=0.046s	t=0.070s	t=0.277s	t=0.282s	t > 5min.	t > 5min.
Sondagem Iterativa	t=0.025s	t=0.116s	t=1.069s	t=28.493s	t > 5min.	t > 5min.

Tabela 1 - Tempos obtidos para as diferentes procuras.

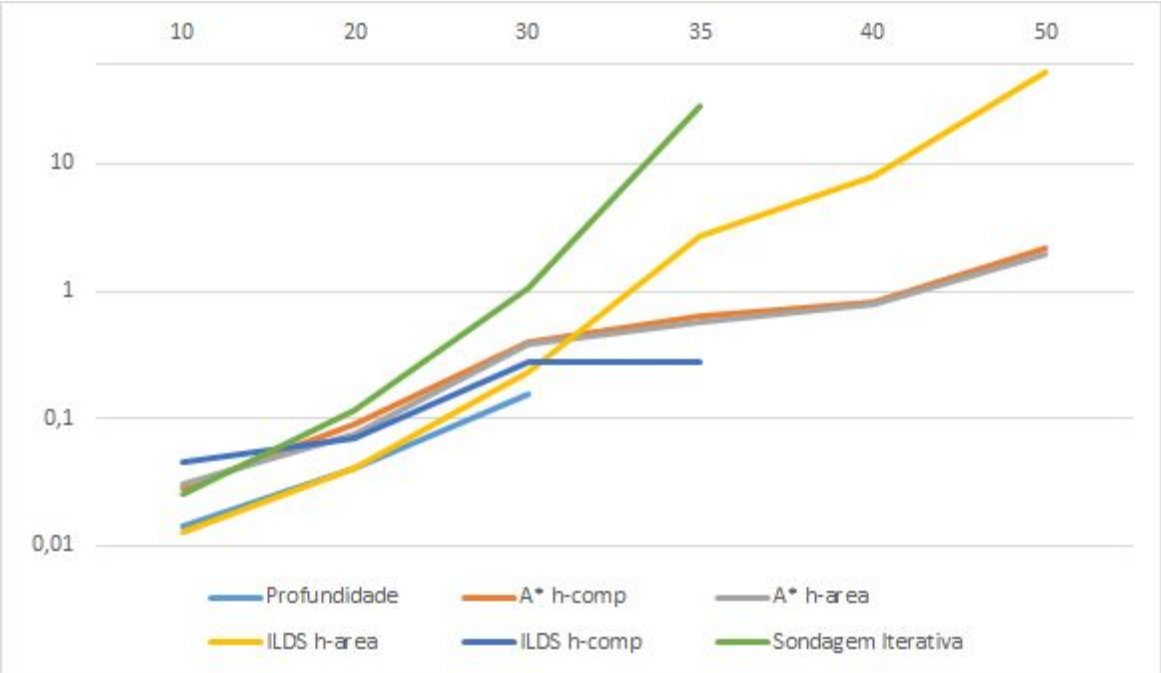

Gráfico 1 - Gráfico com escala logarítmica com os dados da Tabela 1. Valores de tempo superiores a 5 minutos não foram representados. O eixo horizontal representa as peças dos diferentes problemas. O eixo vertical representa o tempo numa escala logarítmica variando entre 0 e 60s.

É claro por observação dos valores obtidos e observáveis no Gráfico 1 que o crescimento temporal para estas procuras é exponencial.

Como vemos, a procura em profundidade tem um crescimento exponencial com uma constante elevada, crescendo o tempo de procura para problemas com peças superiores a 30 para valores superiores a 5 minutos.

A Sondagem Iterativa, baseando-se numa heurística aleatória, é a pior procura observada, visto que os limites dos problemas foram ajustados de forma a diminuir o número de soluções possíveis. Esta procura, como afunda rapidamente, para problemas com inúmeras soluções (mais espaço livre no rectângulo) funcionaria melhor do que os problemas estudados.

Estudando o a procura informada A*, com as heurísticas baseadas na Área e no Comprimento, observa-se que, para os problemas estudados, a heurística da Área obtém resultados ligeiramente melhores. No entanto, como os valores de tempo obtidos são muito próximos, torna-se difícil concluir qual das heurísticas se comporta melhor em geral. Desta forma, decidimos estudar ambas para a procura ILDS.

Para o ILDS com a heurística baseada na Área retornou resposta para todos os problemas, ao contrário da heurística baseada no Comprimento que acaba por crescer exponencialmente em tempo.

Observou-se também uma grande dificuldade em resolver o problema, para casos mais difíceis. Devido a existência de um fator elevadíssimo de ramificação, a solução deve passar pela utilização de uma heurística, que nos guie o melhor possível nesse sentido. Assim sendo, a resolução do problema fica muito dependente da forma como a heurística funciona. Para alguns dos casos testados, em que o espaço de soluções é muito pequeno, as funções heurísticas permitem que os respectivos algoritmos demorem tempos bem diferentes para o mesmo problema. Procurou-se fazer combinações de diversas heurísticas para tentar obter uma que combinasse vantagens de várias mas, apesar de se conseguir fazer isso para alguns casos, a forma de combinação não era suficientemente robusta para casos mais difíceis.

3. Problema de Optimização

3.1. Procura com Sondagem Iterativa

A procura com sondagem iterativa utiliza um algoritmo semelhante ao utilizado para satisfação. Neste caso, não é possível acrescentar melhorias ao algoritmo, pois este não utiliza heurísticas (é uma estratégia não informada). Fez-se apenas questão de chamar a função *samp* múltiplas vezes, até ao fim do tempo.

3.2. Procura Alternativa

A procura alternativa utiliza um algoritmo ILDS adaptado, com algumas optimizações específicas. As funções ILDS-opt e LDS-opt são utilizadas nesta implementação. Por um lado, foi necessário manter a melhor solução encontrada até ao momento, atualizá-la sempre que atingimos um estado objectivo e passá-la como argumento na função. A partir desta, é possível fazer duas optimizações. Por um lado, se a altura atual na pesquisa for superior à menor altura encontrada até agora, podemos retornar na função recursiva. Podemos aproveitar ainda mais este conhecimento para implementar um *bound.* É possível verificar no estado atual, mas considerando o rectângulo obtido com a menor altura atingida até agora, se

ainda é possível colocar qualquer uma das peças que falta. Se uma delas não poder ser adicionada, sem ultrapassar essa altura, então é porque não vai ser possível obter melhor resultado a partir deste estado, e podemos retornar na função recursiva.

Foi ainda necessário verificar se foram excedidos os 5 minutos permitidos para execução. Caso isso tenha acontecido, a função passa a retornar a melhor solução obtida. Para este algoritmo apresentamos os resultados obtidos utilizando duas heurísticas diferentes: h-hole-height (conjuga a heurística baseada em espaços em branco e a heurística baseada em altura).

3.3. Comparação

Para problemas de optimização, como não faz sentido comparar o tempo gasto por cada um, visto que ambos irão correr durante 5 minutos retornando a melhor resposta no final. Desta forma, decidimos comparar então os valores obtidos de altura para cada um dos problemas com um certo tipo de procura. Os problemas usados são os mesmos descritos anteriormente.

	10 peças	**20 peças**	**30 peças**	**35 peças**	**40 peças**	**50 peças**
ILDS (altura + buracos)	10	11	14	16	18	23
ILDS (altura + comprimento)	11	12	14	16	19	24
Sondagem Iterativa	11	12	17	20	22	27

Tabela 2 - Menor altura obtida para as diferentes procuras.

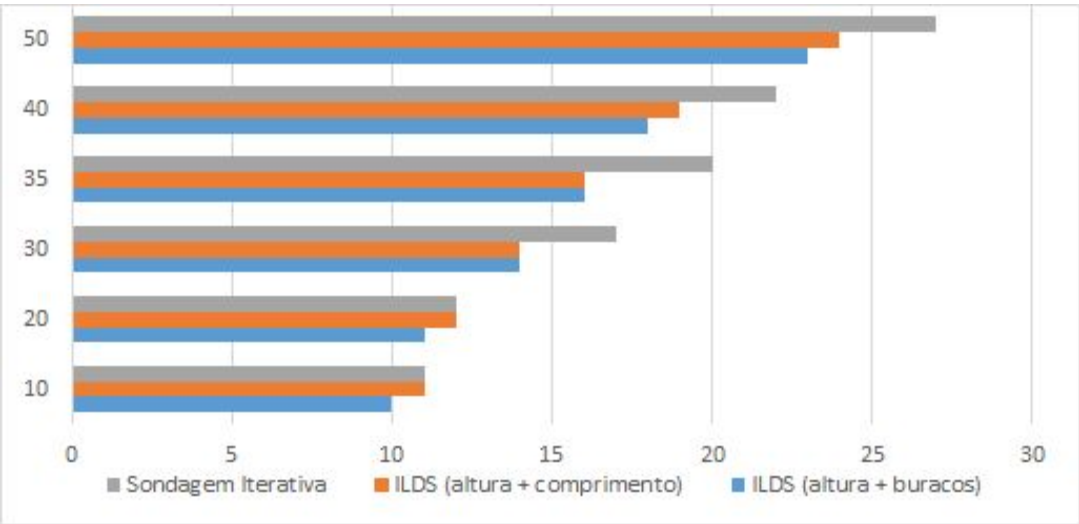

Gráfico 2 - Gráfico com os dados da Tabela 2. O eixo horizontal representa a menor altura encontrada para cada procura. O eixo vertical representa os diferentes problemas analisados.

Os resultados obtidos pela sondagem iterativa não são muito bons, pois como sabemos esta pesquisa é não informada e totalmente aleatória. Quando o espaço de resultados

é suficientemente grande, torna-se muito complicado obter uma solução próxima da ótima, porque se irão gerar várias vezes os mesmos estados, muitos destes piores do que qualquer um obtido.

O ILDS modificado permite obter resultados bem melhores, dependendo da heurística utilizada. Em conjunto com as diversas optimizações consegue-se que o resultado seja bom. A heurística combinada da heurística baseadas na altura e dos buracos é a que apresenta melhores resultados.

4. Conclusão

Para os problemas de satisfação estudados, pode-se concluir que a procura A* apresenta os melhores resultados.

Para as observações feitas, a heurística baseada na Área, em geral, obtém melhores resultados que as restantes, fazendo sentido ser a melhor heurística visto que uma das abordagens de resolver este problema no mundo real seria por os maiores objetos primeiros seguidos de objetos com dimensões inferiores.

A procura ILDS é a melhor alternativa ao A*. Podemos concluir que para este tipo de problema, o melhor tipo de procura são as procuras informadas, visto que apresentaram melhores resultados que as não-informadas.

O crescimento temporal exponencial é esperado para as procuras de profundidade e A*, visto que a sua complexidade temporal é de O(b^d), onde b é o fator de ramificação (que é sempre o número de peças que faltam atribuir * posições disponíveis para atribuir * 2, visto que a orientação pode ser Horizontal ou Vertical) e d é a profundidade (a profundidade é sempre o número de peças que iremos atribuir). Desta forma, é de esperar que a sua complexidade aumente exponencialmente à medida que se aumenta o número de peças no tabuleiro.

Para os problemas de optimização estudados, pode-se concluir que a procura de optimização informada obtém melhores resultados para problemas como estes onde há um limite temporal e o número de peças/variáveis pode ser grande. Como podemos observar pelo Gráfico 2, para problemas com menor número de peças, os resultados não diferem muito mas a optimilidade da procura ILDS é claramente testemunhada à medida que se aumenta o número de peças a colocar.

Sabendo que o objectivo do problema de optimização é encontrar a menor altura possível para um determinado problema, é de esperar que a heurística que minimiza os buracos que possam existir entre as peças, combinada com a heurística de altura, obtenha os melhores resultados, visto que a minimização de buracos ajuda a reduzir a altura (as peças priorizam preencher os espaços mais abaixo o mais eficazmente possível e só depois os mais acima).