Universidade de Coimbra
Faculdade de Ciências e Tecnologia
Departamento de Engenharia Informática

Introdução à Inteligência Artificial
2015/2016 - 2º Semestre

Trabalho Prático Nº2 - Sokoban
Meta Final – Relatório

Alexandra Tomé Leandro	2013146082	PL6
Ana Rita Cabral Dias	2013142131	PL6

Introdução

O *Sokoban é um* jogo do estilo puzzle criado por Hiroyuki Imabayashi em 1981, tendo como objetivo movimentar uma série de caixas para determinadas posições específicas. Estes movimentos estão restringidos por um conjunto de regras:

- O jogador só se pode movimentar uma célula de cada vez na horizontal ou vertical;
- O jogador pode empurrar, mas não puxar as caixas;
- Só pode mover uma caixa de cada vez;
- Não podem existir duas caixas na mesma célula;
- O jogador não se pode sobrepor a caixas;
- O jogador e as caixas não podem transpor as paredes do ambiente do jogo;
- O jogo termina quando todas as caixas estiverem nas posições objetivo definidas.

Assim, o objetivo deste trabalho será o desenvolvimento e implementação de diversos algoritmos de procura que resolvam diferentes problemas de um jogo de *Sokoban*. Este trabalho irá portanto levar à aquisição de competências de modelação de problemas com uma procura num espaço de estados e de competências a nível da elaboração de heurísticas adequadas para problemas específicos. Além disso, irá proporcionar a compreensão de todas as forças, fraquezas e dificuldades inerentes aos diferentes algoritmos de procura abordados. Irá ainda, implicar uma análise empírica comparativa das diferentes abordagens em termos de complexidade temporal e espacial.

Modelação e implementação

Tendo por base os moldes do jogo *Sokoban*, sabemos que um estado será qualquer configuração possível do problema, ou seja, corresponde às posições de todos os elementos do jogo: caixas, jogador e posições destino, no respetivo ambiente do problema. Desta forma, um estado inicial pode ser definido como o mapa carregado inicialmente com as posições do jogador, caixas e objetivos antes de por em execução qualquer algoritmo de procura, isto é, antes de ser feito qualquer movimento. Analogamente, o estado final será aquele que satisfaz o objetivo estipulado pelo problema, a configuração do ambiente onde todas as posições objetivo estiverem preenchidas por uma caixa, o que significa que a posição de cada caixa terá de coincidir com a posição de um dos objetivos.

Sabemos que os operadores de mudança de estado são todas as ações possíveis a partir de um estado específico, permitido atuar sobre o estado atual alterando-o. Neste caso, os operadores serão os movimentos possíveis do jogador: Norte, Sul, Este e Oeste, que irão levar à alteração das posições das caixas. Estes operadores terão de obedecer ao conjunto de regras mencionado anteriormente.

Sendo o estado final, como vimos acima, a configuração do ambiente onde todas as posições dos objetivos correspondem também a posições de caixas, a solução pretendida irá corresponder ao conjunto dos operadores que foram necessários para o estado inicial ser alterado de modo a corresponder ao estado final. Logo, a natureza da solução desejada é o conjunto ordenado dos movimentos que o jogador tem de efetuar para que cada caixa mal posicionada seja movida para uma das posições objetivo.

Considerando o ambiente do problema, e o facto do jogador só poder movimentar-se uma célula de cada vez na horizontal ou vertical, o custo associado a cada movimento corresponderá ao número de células percorridas pelo jogador entre cada transição efetuada, ou seja, a distância matricial entre as posições tomadas.

Para a implementação dos algoritmos de procura foram utilizados métodos de **procura cega** e de **pesquisa informada** com base em heurísticas. Como o nome indica, nos métodos de pesquisa cega, devido à falta de informação sobre o problema, não é possível termos conhecimento dos nós que devem ser testados primeiro, portanto, são utilizadas estratégias que irão definir de um modo sistemático o processo de geração e teste de nós do problema, definindo assim, como a navegação da árvore será efetuada. No caso da pesquisa informada, admitimos que temos alguma informação adicional que nos permite estimar o custo do caminho do nó corrente até ao nó solução, este valor será obtido pela função *h(n),*que nos permite escolher de forma mais inteligente os próximos nós a serem analisados.

Dentro da pesquisa cega foram implementados e analisados três algoritmos distintos:

1. **Largura Primeiro:** esta estratégia caracteriza-se por analisar todos os nós de um determinado nível antes dos nós do nível seguinte até encontrar uma solução, ou seja, os nós de um dado nível serão gerados todos na mesma iteração.
   Assim, para a implementação deste algoritmo é utilizado uma fila, onde serão guardados os elementos a analisar. O próximo nó a ser expandido será sempre primeiro nó da fila, e, no caso de não ser a solução pretendida, os seus sucessores são adicionados no final da fila, obtendo assim uma estrutura onde os primeiros elementos guardados são os primeiros a sair (FIFO).

2. **Pesquisa em profundidade limitada:** a limitação associada a este algoritmo implica que, os operadores só serão aplicados se o nível do nó atual a ser testado for inferior ao máximo estipulado, fixando o limite máximo de procura. Além deste fator, o algoritmo consiste em expandir um certo nó, escolher um dos seus sucessores, expandir esse sucessor continuando o processo até encontrar uma solução ou até não existirem mais nós para expandir. Neste último caso, o processo é continuado por um nó irmão do último analisado, se existir, ou irá regressa ao nível anterior para continuar a procura. Sendo que durante todo o processo é verificado, para cada estado, se o seu nível não ultrapassa o limite.
   Esta estratégia distingue-se da anterior principalmente na estrutura escolhida para guardar a fronteira da árvore de procura, neste caso, uma pilha. Desta forma, o primeiro elemento guardado será o primeiro a ser testado e expandido, e os seus sucessores serão guardados também no início da pilha (LIFO), o que permite uma pesquisa em profundidade na árvore da procura.

3. **Aprofundamento progressivo**: o algoritmo em questão é baseado no anterior, com uma adaptação para auxiliar nos casos mais genéricos. Nomeadamente, quando não existe informação suficiente para estabelecer um limite máximo adequado onde se poderá encontrar solução e nos casos em que o espaço de procura é relativamente grande.
   A implementação consiste em executar repetidamente procuras de profundidade limitada começando com o limite a zero e incrementando-o até encontrar solução. Para isso temos novamente como estrutura uma pilha, que irá ser preenchida como na estratégia anterior até a um determinado limite. Despois dessa execução, se a pilha estiver vazia e ainda não tiver sido encontrada nenhuma solução, o algoritmo de procura em profundidade limitada terá de ser reiniciado, desta vez, com um limite superior ao usado antes. Portanto, é necessário colocar novamente a raiz da árvore de procura, o estado inicial, no início da pilha para recomeçar o processo. Este ciclo é repetido até ser encontrada uma solução.

Dentro dos métodos de procura heurística, foram desenvolvidos e estudados as seguintes estratégias:

1. **Pesquisa sôfrega:** para este algoritmo, o estado escolhido para ser analisado primeiro será o nó na fronteira da árvore de procura que aparenta ser o mais promissor de acordo com o valor estimado por *h(n)*, que será uma estimativa do custo total de transitar do estado atual até ao final. O método mantém então, a fronteira da árvore de procura ordenada crescentemente de acordo com os valores calculados pela função *h(n). O* próximo nó a ser testado e expandido será sempre aquele com menor valor de *h*, o que hipoteticamente estará mais próximo da solução.
   Na programação desta estratégia é utilizada necessariamente uma lista ordenada, para tal, foi elaborada um algoritmo de ordenamento por inserção, que, quando existe um novo nó a inserir, irá comparar o valor do seu custo estimado com os valores de *h* dos estados da lista atual e colocá-lo na posição correta. Assim, esta estratégia irá expandir o primeiro elemento da lista e guardar os respetivos sucessores de forma ordenada e continuar o processo até encontrar solução desejada.

2. **A*:** neste caso é feita uma junção do critério anterior (função *h(n))*, com o utilizado no algoritmo de custo uniforme, *g(n)*, que representa o custo para chegar do nó inicial ao nó atual. A função utilizada como critério de ordenação passa a ser a soma das duas funções: *f(h)=g(n)+h(n).* Sabemos que *h(n)* nos dá um valor estimado do custo efetivo, portanto *f(n)* também quantifica o valor estimado até ao estado final passando pelo estado n.
   Em termos de implementação, este algoritmo é semelhante ao anterior, com a diferença que, para além da heurística calculada em *h(n)*, cada nó irá ter também um parâmetro correspondente ao resultado do cálculo da função *g(n)*, sendo que a soma das duas funções será a o critério para ordenar a lista onde são guardados os estados a explorar.

A performance de ambas abordagens estará dependente da qualidade da heurística escolhida e criada. Para ser admissível, o valor calculado pela função nunca poderá ser superior ao custo real da transição em causa. Inicialmente, a heurística elaborada corresponde ao número de caixas que não estão na posição destino, assume-se que é necessário, no mínimo, um movimento para deslocar cada caixa para uma das posições objetivo. Como vamos ver mais à frente, embora admissível, pois o valor calculado é inferior ao real, esta heurística subestima demasiado o custo das transições, logo não apresenta grande melhoramento em relação aos métodos de procura cega.

Refinamento

As heurísticas, como vimos, implicam um conhecimento do problema que auxilie na travessia do espaço de procura. Uma heurística pode ser descrita como uma versão menos restringida do problema a explorar. Uma boa heurística, para além de admissível, deve fazer uma estimativa do custo o mais próxima possível do valor real, sem o ultrapassar, sendo que quanto mais próximo do valor verdadeiro, melhor será a heurística, e consequentemente mais eficiente o algoritmo de procura.

1) Heurísticas Idealizadas:

Com o objetivo de melhorar significativamente os métodos de procura informada em relação aos métodos de pesquisa cega desenvolvemos várias heurísticas:

(0) difCaixa() – nesta heurística descrita no enunciado, a estimativa de custo de transitar de um estado s para o estado final é igual ao número de caixas que não estão numa posição destino, estamos assim a assumir que é necessário um movimento para deslocar uma caixa mal posicionada para uma posição objetivo. Obviamente, esta heurística será admissível, pois nunca poderá ultrapassar o valor do custo real necessário para deslocar todas as caixas para posições destino. No entanto, tendo em conta o problema, podemos perceber que o valor estimado será muito inferior ao real, pois não contabiliza a distância que o jogador terá de percorrer até às caixas não posicionadas. Portanto, apesar de admissível não será uma boa heurística, como iremos comprovar pelos resultados da experimentação mais abaixo.

(A) crateMaxDist() – esta estimativa devolve a distância máxima entre o jogador e uma caixa fora da posição. O valor estimado será sempre menor ou igual ao custo real ate à solução, já que no mínimo o *player* desloca-se até à caixa mais distante.

(B) crateTotalDist() – semelhante ao anterior, a estimativa é calculada com a distância entre o jogador e as caixas fora da posição, mas neste caso é devolvido o total de todas as distâncias. Não será admissível em alguns casos onde não é necessário percorrer tanta distância entre caixas.

(C) goalsDistMax() – Aqui verificamos as posições destino que ainda não foram preenchidas e é devolvida a distância máxima entre essas posições objetivo e o jogador. Será admissível, uma vez que será sempre percorrido no mínimo a distância até a posição objetivo mais afastada.

(D) goalsTotalDist() – neste caso, a estimativa calculada baseia-se na soma das distâncias calculadas entre o jogador e as posições objetivo que ainda não foram preenchidas. De forma semelhante à crateTotalDist() poderá ser não admissível.

(E) crateMaxGoalsMed() – esta função de heurística será o resultado da distância máxima entre o jogador e as caixas não posicionadas, e a distancia mínima entre uma posição objetivo não preenchida e uma caixa não posicionada multiplicado pelo número caixas que não estão numa posição objetivo. Aqui, assumimos que será no mínimo necessário percorrer a distância mínima para uma caixa ser deslocada até à posição *goal*, para além de ser necessário uma deslocação do jogador até à caixa.

(F) crateGoalDist() – para esta função a estimativa será a junção de duas outras estimativas anteriores, é calculada a distância máxima entre uma posição destino vazia e uma caixa não posicionada com a distância máxima entre uma caixa e um jogador.

(G) remainingMaxcrates() – neste caso o valor estimado obtido é a soma da distância máxima entre um jogador e uma caixa não posicionada, e o número de caixas que ainda está fora do sítio.

Além destas heurísticas foram ainda implementadas outras com performance muito baixas ou que sobrestimavam o valor em grande margem (não sendo portanto admissíveis). Por exemplo, o cálculo do valor total das distâncias entre o jogador e caixas não posicionadas somado com o número de caixas que não estão bem posicionadas e a soma total de distâncias entre o jogador e as caixas e entre as caixas e as posições destino. Estas sobrestimavam muito o custo e por isso não eram admissíveis, no entanto podem ser consideradas para uso nos casos

em que melhor performance temporal é mais importante do que encontrar o caminho mais curto.

Ainda, experimentámos diversas versões que calculavam as distâncias mínimas entre jogador e as caixas não posicionadas ou distância mínima entre as posições objetivo vazias e o jogador e verificámos que ficavam muito abaixo do custo real, não sendo portanto boas heurísticas.

2) Cálculo das distâncias nas heurísticas: Manhattan versus Euclidiana

Manhattan: É padrão em movimentos de 4 direções, como é o caso. Como o nosso custo mínimo é 1, pode ser definida como: $|p_x - q_x| + |p_y - q_y|$

Euclidiana: Distância em linha reta usada quando existem movimentos em qualquer angulo (em vez das 4 direções). É mais curta que a anterior e neste caso, pode ser definida como:

$$\sqrt{(p_x - q_x)^2 + (py - qy)^2)}$$

Assim, é esperado que a distância Manhattan nos dê custo mais próximo ao custo real e que por isso seja mais eficiente.

A fim de escolher a distância a utilizar efetuámos comparações entre estas duas fórmulas em todas as heurísticas definidas anteriormente. Foi usado o *mapa03* com algoritmo de procura A*.

Distância	A	B	C	D	E	F	G
Manhattan	Visitados: 89437 Expandidos: 251706 Caminho:29	Visitados: 8432 Expandidos: 24132 Caminho:29	Visitados: 90923 Expandidos: 255560 Caminho:29	Visitados: 4104 Expandidos: 11324 Caminho: 31	Visitados: 89437 Expandidos: 251706 Caminho: 29	Visitados: 56219 Expandidos: 154732 Caminho:29	Visitados: 54286 Expandidos: 154557 Caminho:29
Euclidiana	Visitados: 105361 Expandidos: 301610 Caminho:29	Visitados: 13777 Expandidos: 39310 Caminho:29	Visitados: 109486 Expandidos: 311489 Caminho:29	Visitados: 10531 Expandidos: 301610 Caminho: 29	Visitados: 89732 Expandidos: 254639 Caminho:29	Visitados: 104634 Expandidos: 298133 Caminho:29	Visitados: 65610 Expandidos: 189457 Caminho:29

Como esperado, a distância de Manhattan tem melhores resultados neste ambiente, uma vez que se aproxima mais do custo real. Salienta-se que, segundo os resultados, se pode concluir desde já que uma das heurísticas previstas como não admissível o é de facto (o caminho solução encontrado é de tamanho 31 enquanto que nas heurísticas admissíveis é 29).

3) Heurísticas Utilizadas:

De acordo com os resultados anteriores iremos utilizar em maior escala a distância de Manhattan nas heurísticas, no entanto, iremos utilizar os dois métodos de cálculo de distâncias em duas heurísticas, por termo de comparação. São elas:

(o) difCaixa()

(A) crateMaxDist()

(B) crateTotalDist()

(C) M_goalsDistMax() –Distância de Manhattan

(D) E_goalsDistMax() – Distância Euclidiana

(E) goalsTotalDist()

(F) crateMaxGoalsMed()

(G) M_crateGoalDist() – Distância de Manhattan

(H) E_createGoalDist() - Distância Euclidiana

(I) remainingMaxcrates()

Experimentação

De modo a testar e analisar os algoritmos, nomeadamente as suas complexidades temporais e espaciais, foram elaborados e utilizados vários mapas com diferentes dimensões e complexidades, *figura 1* a *figura 8*.

Como iremos ver posteriormente, não foi possível testar alguns destes mapas para todos os algoritmos devido, não só à complexidade dos métodos usados, mas também por fatores dependentes das máquinas utilizadas. Por vezes o *Unity* bloqueava, deixando de responder o que tornou impossível a obtenção de resultados para alguns casos.

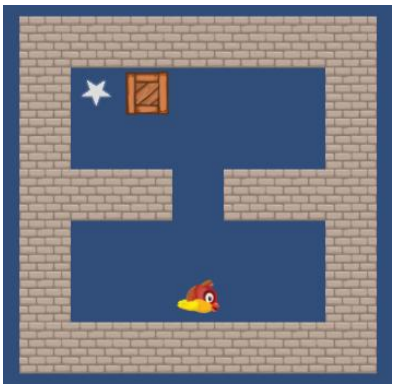

Figura 1 - Mapa01

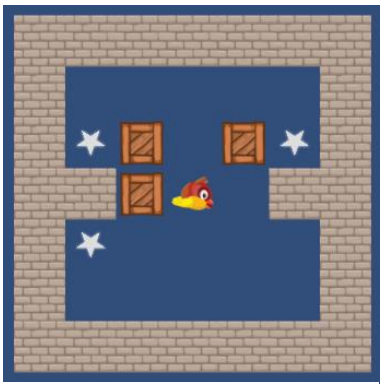

Figura 2 - Mapa02

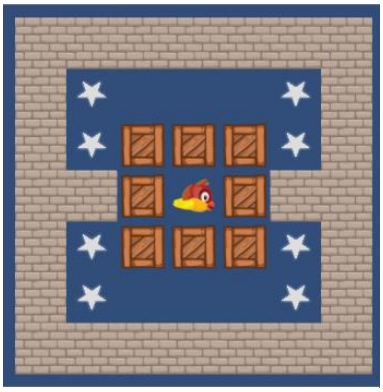

Figura 3 - Mapa03

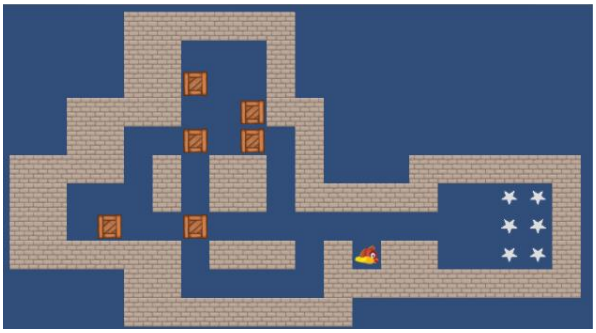

Figura 4 - Mapa04

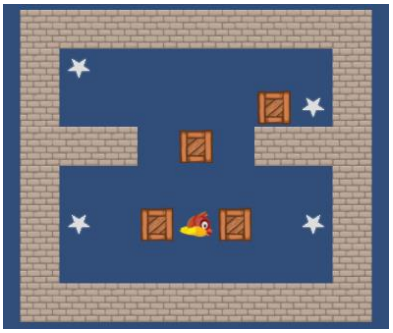

Figura 5 - Mapa05

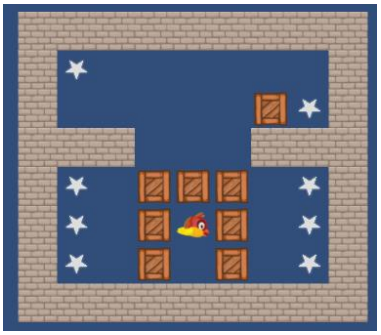

Figura 6 - Mapa06

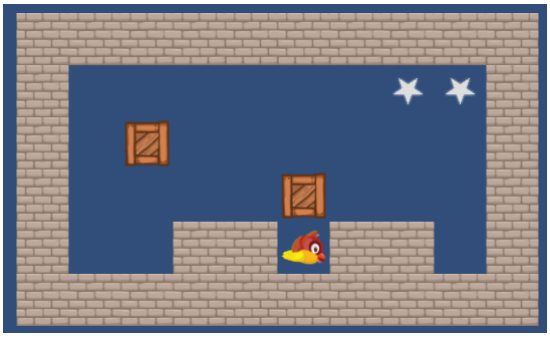

Figura 7 - Mapa07

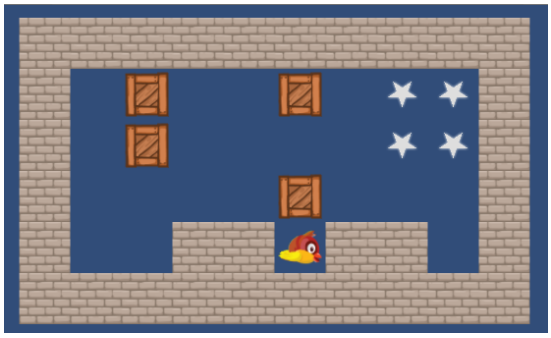

Figura 8 - Mapa08

Cada um destes mapas foi testado para os algoritmos de pesquisa elaborados e as heurísticas acima mencionadas. De seguida, apresentamos as tabelas de cada algoritmo com os valores obtidos para: número de nós visitados, número de nós expandidos e tamanho do caminho da solução encontrada. Sendo que os nós visitados dão informação sobre a **complexidade temporal** do algoritmo, os nós expandidos aludem à **complexidade espacial** e o tamanho do caminho permite determinar se a solução encontrada é a melhor, isto é, se o algoritmo é **discriminatório**.

Nas tabelas abaixo, os campos assinalados com **(*)** representam as situações em que não foi possível obter resultados devido às dimensões e complexidade dos mapas, ao comportamento exponencial a nível temporal de todos os algoritmos abordados e/ou problemas de performance associados aos computadores utilizados.

Largura Primeiro

mapa01	mapa02	mapa03	mapa04	mapa05	mapa06	mapa07	map08
Visitados: 22 Expandidos: 58 Caminho: 5	Visitados: 386 Expandidos: 1123 Caminho: 8	Visitados: 134251 Expandidos: 382032 Caminho: 29	(*)	Visitados: 808261 Expandidos: 2539930 Caminho: 18	(*)	Visitados:75650 Expandidos:222341 Caminho:21	Visitados: 6551195 Expandidos:18732782 Caminho: 38

Sendo que nesta estratégia os nós são analisados por nível, há sempre a possibilidade de surgirem estados já previamente testados e expandidos, o que leva à possibilidade da existência de ciclos, para além de tornar o algoritmo muito mais lento. Ainda assim, todos os nós de cada nível da árvore de procura serão analisados e testados pelo algoritmo, portanto irá sempre ser encontrada uma das soluções existentes, logo o algoritmo é **completo**. Como não é garantido que o custo das transições entre os estados nos níveis seja crescente, não se pode garantir que o algoritmo seja discriminatório. Teoricamente, em termos de complexidade espacial e temporal será $O(r^n)$, onde *n* corresponde ao nível onde a solução do problema é encontrada e *r* é o fator de ramificação, que será o numero de sucessores de um nó. Assim podemos considerar que o fator de ramificação será no máximo 4 (Norte, Sul, Este, Oeste) e o nível de solução corresponderá aproximadamente ao tamanho do caminho encontrado. Verificámos que, com o aumento das dimensões do mapa e com um número maior de caixas a mover, a complexidade aumenta com um comportamento exponencial, como seria de esperar. Sendo que para mapas muito complexos, como *mapa04 e mapa06,* não é sequer possível encontrar em tempo útil uma solução para os problemas, pois o tempo necessário excede o que se poderia considerar viável para uma operação de procura.

Profundidade limitada

mapa01	mapa02	mapa03	mapa04	mapa05	mapa06	mapa07	map08
Visitados: 21 Expandidos: 58 Caminho:5 Limite mínimo: 6	Visitados: 92 Expandidos: 58 Caminho: 10 Limite mínimo: 11	Visitados: 2694 Expandidos:7642 Caminho: 45 Limite mínimo: 46	(*)	Visitados: 244012 Expandidos: 786680 Caminho:45 Limite mínimo: 46	(*)	Visitados:15119 Expandidos:46646 Caminho:62 Limite mínimo:101	Visitados: 333033 Expandidos:998467 Caminho: 165 Limite mínimo: 166

Este algoritmo, embora seja um método de procura cega, implica que haja algum conhecimento para que se possa deduzir em que nível a solução se deve encontrar. Se o limite escolhido não for adequado ao problema em questão, poderá não ser possível encontrar solução, logo o algoritmo não será completo. Foram testados diversos limites para cada um dos ambientes, e verificou-se que, nos casos em que o limite escolhido ultrapassa o limite mínimo

necessário para encontrar solução, o comprimento da solução obtida é, por norma, superior ao caminho obtido anteriormente. Isto demonstra que o algoritmo **não é discriminatório.** Esta estratégia é baseada na procura em profundidade, mas devido à limitação do nível é evitado o problema dos ciclos ou caminhos infinitos. De acordo com os valores teóricos a complexidade temporal será, semelhante à estratégia anterior $O(r^n)$ e a complexidade espacial é O(l*r), onde *l* é o limite do nível máximo fixado. Podemos ver que, comparativamente aos valores da pesquisa em largura primeiro, o número de nós expandidos e visitados obtido é muito inferior, para os limites mais próximos da solução. Ainda assim, nos mapas 4 e 6 não é possível encontrar solução em tempo útil.

Aprofundamento progressivo:

mapa01	mapa02	mapa03	mapa04	mapa05	mapa06	mapa07	map08
Visitados: 61 Expandidos: 172 Caminho: 5	Visitados:1260 Expandidos: 3753 Caminho:10	Visitados:132001 Expandidos: 354695 Caminho: 35	(*)	Visitados:623598 Expandidos:2025528 Caminho:36	(*)	Visitados:116598 Expandidos:363546 Caminho:48	(*)

O aprofundamento progressivo resolve o problema da possível falta de conhecimento para estabelecer um limite de nível máximo na profundidade limitada. No entanto, este algoritmo é bastante mais complexo espacial e temporalmente em relação ao anterior, pois implica a construção e reconstrução da árvore de procura para cada nível máximo fixado até ser encontrada uma solução, o que o torna um algoritmo muito pesado para mapas maiores onde a solução estará a níveis mais elevados. Sendo um algoritmo de profundidade nem sempre obterá o melhor caminho possível, logo, **não é discriminador**, como podemos ver pelo tamanho dos caminhos encontrados que nem sempre é o menor possível. Como o algoritmo implica a execução de uma pesquisa em profundidade em ciclo com incremento do limite até ser encontrada uma solução, será sempre **completo**. Em termos teóricos a complexidade espacial será de $O(r^n)$ e a sobrecarga da complexidade temporal, como os nós são analisados mais do que uma vez, será da ordem de $\frac{r}{r-1}$. Pelos resultados obtidos podemos comprovar empiricamente esses valores de complexidade que aumentam em rácio relativamente aos da profundidade limitada de acordo com as dimensões e complexidade do mapa.

Sôfrega:

Heurística	mapa01	mapa02	mapa03	mapa04	mapa05	mapa06	mapa07	map08
difCaixa	Visitados:19 Expandidos:50 Caminho:5	Visitados:30 Expandidos:96 Caminho:8	Visitados:3506 Expandidos:9568 Caminho:37	(*)	Visitados:839 Expandidos:2627 Caminho:24	Visitados:1328 Expandidos:4316 Caminho:42	Visitados:4739 Expandidos:14155 Caminho:21	Visitados:5659 Expandidos:1607 Caminho:40
crateMaxDist	Visitados:5 Expandidos:16 Caminho:5	Visitados: 21 Expandidos: 70 Caminho: 8	Visitados:2944 Expandidos: 8374 Caminho:37	(*)	Visitados:30697 Expandidos:15292 Caminho:30	(*)	Visitados:1221 Expandidos:3899 Caminho:31	Visitados:1654 Expandidos:5223 Caminho:52
crateTotalDist	Visitados:5 Expandidos:16 Caminho:5	Visitados: 20 Expandidos: 67 Caminho: 10	Visitados:1127 Expandidos:3172 Caminho:41	(*)	Visitados: 124 Expandidos:443 Caminho:24	Visitados:692 Expandidos:2419 Caminho:62	Visitados:133 Expandidos:423 Caminho:23	Visitados:1670 Expandidos:5229 Caminho:61
M_goalsDistMax	Visitados: 5 Expandidos: 16 Caminho:5	Visitados: 49 Expandidos: 156 Caminho:8	Visitados: 12838 Expandidos:36047 Caminho:35	(*)	(*)	(*)	Visitados:3784 Expandidos:11386 Caminho:22	Visitados:104802 Expandidos:306372 Caminho:54
E_goalsDistMax	Visitados:13 Expandidos:35 Caminho:5	Visitados:323 Expandidos:997 Caminho:8	Visitados:1487 Expandidos:4067 Caminho:35	(*)	(*)	(*)	Visitados:3427 Expandidos:10350 Caminho:22	Visitados:51314 Expandidos:148508 Caminho:38
goalsTotalDist	Visitados:5 Expandidos: 16 Caminho: 5	Visitados:63 Expandidos:197 Caminho: 10	Visitados:1487 Expandidos:4067 Caminho:35	(*)	Visitados:1323 Expandidos:4224 Caminho:26	Visitados:673 Expandidos:2122 Caminho:48	Visitados:942 Expandidos:2877 Caminho:22	Visitados:171790 Expandidos:501223 Caminho:38
crateMaxGoalsMed	Visitados:5 Expandidos:16 Caminho:5	Visitados:21 Expandidos:70 Caminho:8	Visitados:2944 Expandidos: 8374 Caminho:37	(*)	Visitados:30697 Expandidos:105292 Caminho:30	(*)	Visitados:1221 Expandidos:3899 Caminho:31	Visitados:1654 Expandidos:5223 Caminho:52
M_crateGoalDist	Visitados:5 Expandidos: 16 Caminho: 5	Visitados:51 Expandidos:157 Caminho:8	Visitados: 1618 Expandidos:4676 Caminho:37	(*)	Visitados:5068 Expandidos:17148 Caminho:44	(*)	Visitados:185 Expandidos:604 Caminho:23	Visitados:565 Expandidos:1778 Caminho:52
E_crateGoalDist	Visitados: 5 Expandidos: 16 Caminho: 5	Visitados:25 Expandidos:81 Caminho:8	Visitados: 1100 Expandidos:3108 Caminho:37	(*)	Visitados:3637 Expandidos:12125 Caminho:48	(*)	Visitados:449 Expandidos:1461 Caminho:36	Visitados:4691 Expandidos:13884 Caminho:62
remainingMaxcrates	Visitados:5 Expandidos: 16 Caminho: 5	Visitados:25 Expandidos:81 Caminho:8	Visitados:1100 Expandidos:3108 Caminho:37	(*)	Visitados:358 Expandidos:1242 Caminho:22	Visitados:3877 Expandidos:12005 Caminho:47	Visitados:211 Expandidos:689 Caminho:23	Visitados:979 Expandidos:3076 Caminho:52

Este algoritmo depende fundamentalmente da função heurística utilizada. De maneira geral, o algoritmo pode não ser completo pois há a possibilidade de surgirem caminhos infinitos, quando nós já expandidos são novamente analisados. Além disso, também **não é discriminatório**, em norma, pois não tem em conta o custo associado da transição entre o estado inicial e o atual, apenas ordena os nós em função da estimativa do custo total de transição entre o estado atual e o estado final. Mais uma vez, as complexidades temporais e espaciais serão hipoteticamente $O(r^n)$, mas podem ser melhoradas dependendo da função heurística escolhida. Assim, para boas heurísticas, aquelas que se aproximam mais do valor do custo real, como *remainingMaxcrates()*, observamos que os valores de nós expandidos e visitados são inferiores em relação aos resultados das outras heurísticas admissíveis utilizadas. Vemos também que, para heurísticas mais simples, que subestimam em demasia o valor do custo real, como *difCaixa()* e as heurísticas com distâncias euclidianas, os resultados não diferem muito daqueles obtidos com os algoritmos de procura cega.

Evidencia-se que, os resultados em mapas simples como o primeiro não sofrem muitas alterações de heurística para heurística, uma vez que o mapa não é muito grande (diversos caminhos possíveis) nem muito complexo e por isso não é difícil encontrar um caminho solução.

É ainda de salientar que, no caso das funções de cálculo dos totais das distâncias, o número de nós visitados e expandidos é bastante inferior aos das restantes heurísticas, isto deve-se ao facto de não serem admissíveis, ou seja, a estimativa calculada pode ser superior ao valor real. Aliás, em alguns mapas só são obtidos os caminhos em tempo útil nestas heurísticas.

A*:

Heurística	mapa01	mapa02	mapa03	mapa04	mapa05	mapa06	mapa07	map08
difCaixa	Visitados:21 Expandidos:55 Caminho:5	Visitados:187 Expandidos:564 Caminho:8	Visitados:126745 Expandidos:362926 Caminho: 29	(*)	Visitados:194152 Expandidos:614984 Caminho: 18	(*)	Visitados: 60935 Expandidos:179874 Caminho:21	(*)
crateMaxDist	Visitados:9 Expandidos:27 Caminho:5	Visitados:92 Expandidos:290 Caminho:8	Visitados:89437 Expandidos:251706 Caminho:29	(*)	Visitados:59321 Expandidos:200727 Caminho:18	(*)	Visitados:22040 Expandidos:68743 Caminho:21	(*)
crateTotalDist	Visitados:9 Expandidos: 27 Caminho:5	Visitados:32 Expandidos:105 Caminho:8	Visitados:8432 Expandidos:24132 Caminho:29	(*)	Visitados:1871 Expandidos:6603 Caminho:18	(*)	Visitados:9088 Expandidos:28544 Caminho:21	Visitados:264166 Expandidos:810347 Caminho:38
M_goalsDistMax	Visitados:12 Expandidos: 33 Caminho:5	Visitados:97 Expandidos:309 Caminho:8	Visitados:90923 Expandidos:255560 Caminho:29	(*)	Visitados:41811 Expandidos:143225 Caminho:18	(*)	Visitados:29382 Expandidos:88377 Caminho:21	(*)

E_goalsDistMax	Visitados:13 Expandidos:36 Caminho:5	Visitados:124 Expandidos:379 Caminho:8	Visitados: 109486 Expandidos:311489 Caminho: 29	(*)	Visitados:106693 Expandidos: 362903 Caminho:18	(*)	Visitados: 37924 Expandidos:114607 Caminho:21	(*)
goalsTotalDist	Visitados:12 Expandidos:33 Caminho:5	Visitados:79 Expandidos:246 Caminho:10	Visitados:4104 Expandidos:11324 Caminho:31	(*)	Visitados: 166 Expandidos:526 Caminho:18	(*)	Visitados:3766 Expandidos:11710 Caminho:21	(*)
crateMaxGoalsMed	Visitados:9 Expandidos: 27 Caminho:5	Visitados:92 Expandidos:290 Caminho:8	Visitados:89437 Expandidos:251706 Caminho:29	(*)	Visitados:59321 Expandidos:200727 Caminho:18	(*)	Visitados: 22040 Expandidos:68743 Caminho:21	(*)
M_crateGoalDist	Visitados: 8 Expandidos:25 Caminho:5	Visitados:92 Expandidos:281 Caminho:8	Visitados:56219 Expandidos:154732 Caminho:29	(*)	Visitados:61784 Expandidos:205496 Caminho:18	(*)	Visitados:2928 Expandidos:9287 Caminho:21	(*)
E_crateGoalDist	Visitados:7 Expandidos: 22 Caminho: 5	Visitados:118 Expandidos:366 Caminho:8	Visitados:104634 Expandidos:298133 Caminho:29	(*)	Visitados:131261 Expandidos:441415 Caminho:18	(*)	Visitados:4943 Expandidos:15282 Caminho:21	(*)
remainingMaxcrates	Visitados:8 Expandidos: 25 Caminho:5	Visitados:71 Expandidos:221 Caminho:8	Visitados:54286 Expandidos:154557 Caminho:29	(*)	Visitados:13830 Expandidos:47133 Caminho:18	(*)	Visitados:16887 Expandidos:52799 Caminho:21	(*)

Analogamente à pesquisa sôfrega, a performance da estratégia A* estará relacionada com a qualidade da heurística escolhida. O algoritmo será sempre **completo**, pois o fator de ramificação de cada nó é finito e o custo total das transições calculadas nunca poderá ser negativo. Além disso, sendo a heurística utilizada admissível sabemos que a solução obtida será ótima, ou seja, o algoritmo é **discriminatório**, porque, para além da função *h(n)* que subestima o valor do custo de transitar do estado atual até ao estado final, temos também em conta o custo total entre o estado inicial e o atual, obtendo assim sempre o caminho mais curto possível.

Como seria de esperar, as heurísticas baseadas na soma de várias distâncias não são admissíveis, pois não é garantido que o jogador tenha de percorrer todos esses caminhos (caixas e posições objetivo podem estar perto uma das outras). Assim, o valor estimado por estas funções – *goalsTotalDist()* e *cratesTotal()* - pode, em certos ambientes, ser superior ao custo real. Podemos comprovar isso na heurística *goalsTotalDist*() no caso do mapa, onde o caminho encontrado é de tamanho 31, quando a solução ótima tem comprimento 29, se a solução encontrada não é ótima a função de heurística associada não será admissível.

No entanto, apesar de o caminho ser maior, os valores obtidos com o uso destas heurísticas (complexidade temporal e espacial) são bastante inferiores aos das outras funções implementadas, tornando o algoritmo de procura mais rápido.

Por outro lado, as funções mais simples, revelaram não ser muito eficientes em termos temporais ou espaciais, pois a heurística usada é muito inferior ao custo real, não auxiliando significativamente a navegação na árvore de procura. Portanto, embora as heurísticas sejam admissíveis, em termos de performance, os algoritmos comportam-se de forma semelhante aos algoritmos de procura cega. Dentro deste grupo de funções temos: E_*goalsDistMax()*, *difCaixa()* e E_*crateGoalDist()*.Desta forma, concluímos que a heurística com melhor comportamento dentro das admissíveis será a *remainingMaxcrates()*, já que é a que apresenta menores complexidades temporais e espaciais.

Conclusões:

1) Embora o algoritmo de Largura seja mais lento e não discriminatório, é interessante porque uma vez que analisa todos os nós chega sempre a um caminho se este existir.

2) O algoritmo de Profundidade com limite será útil se se souber definir limite ideal, mas este é muito difícil de encontrar para mapas grandes e complexos. Caso usemos limite abaixo do ideal, a solução não será encontrada e portanto o algoritmo não é completo, e caso usemos um limite acima as complexidades e o tamanho do caminho encontrado serão maiores.

3) Pesquisa com Aprofundamento Progressivo permite corrigir o facto do limite escolhido na estratégia de profundidade limitada ser fulcral para o seu funcionamento, mas é incomportável para mapas muito complexos ou de grandes dimensões, uma vez que é bastante lento porque testa todos limites até encontrar uma solução.

4) O algoritmo de procura Sôfrega tem melhor comportamento em relação aos de pesquisa cega mas pode não ser completo e o seu comportamento depende muito da escolha de heurística, o que é uma tarefa difícil.

5) As heurísticas não admissíveis permitem obter uma solução mais rapidamente, ou seja, com menor complexidade temporal, mas nem sempre devolvem caminhos pequenos. Ou seja, devem ser usados quando o tamanho do caminho não é muito relevante, e a necessidade de resultados mais rápidos prevalece, por exemplo, em mapas muito grandes e complexos.

6) As heurísticas admissíveis são, regra geral, mais lentas que as não admissíveis, mas encontram, por norma, caminhos menores.

7) O método de procura A* é discriminatório quando as heurísticas aplicadas são admissíveis, ou seja, retorna o menor caminho possível. No entanto, é mais lento que a pesquisa sôfrega.

8) Quando os mapas são simples, a escolha de heurísticas nos algoritmos de procura Sôfrega e A* não são muito importantes, porém, quando o mapa é de maiores dimensões e mais complexo, estas escolhas fazem a diferença.

9) Nem sempre os algoritmos devolvem resultados em tempos úteis, chegando a estar horas sem devolverem qualquer caminho solução. Quanto maior ou mais complexo o mapa (e dependendo do algoritmo/heurística utilizada) maior será o tempo de execução.

10) Nas procuras Sôfrega e A* deve ser usado um algoritmo de ordenação de nós rápido, de forma a aumentar a performance dos métodos de pesquisa e obter execuções em tempo útil.