Universidade de Coimbra

Faculdade de Ciências e Tecnologia

Departamento de Engenharia Informática

Sokoban

Introdução à Inteligência Artificial

Relatório

Christopher Liu, Nº 2013150914

Emanuel Matos, Nº 2011149813

Guilherme Peixoto, Nº 2010138049

Intrudução

O segundo trabalho prático da disciplina de Introdução à Inteligência Artificial incidiu sobre agentes de procura no jogo Sokoban. O trabalho foi realizado novamente em Unity utilizando a linguagem C#.

Neste trabalho, partindo do código fornecido pelos docentes que já continha um algoritmo de procura em largura, adicionamos dois algoritmos de pesquisa cega, profundidade limitada e aprofundamento progressivo. Também desenvolvemos dois algoritmos de pesquisa informada, A* e pesquisa sôfrega. Nestes dois últimos algoritmos podemos utilizar três heurísticas diferentes, uma fornecida pelo docente e doas que desenvolvemos.

Posto isto, começamos a testar os algoritmos nos doze mapas fornecidos e a análise dos mesmos.

Modelação

O Sokoban é um jogo em que o jogador tem de movimentar um determinado número de caixas até uma posição indicada no mapa, o jogo é dado por terminado quando todas as caixas estão nas posições finais.

Este jogo deve obedecer a algumas regras tais como:

- Não é permitido ao jogador empurrar mais do que uma caixa por movimento;
- O jogador só pode fazer movimentos horizontais e verticais, nunca na diagonal, e só se desloca uma célula de cada vez;
- O jogador não pode estar na mesma célula que uma caixa bem como duas caixas não podem estar na mesma célula;
- As paredes não podem ser transpostas pelo jogador e pelas caixas, são obstáculos que delimitam o mapa.

No nosso trabalho o jogador é representado por um pássaro e as posições finais por estrelas.

Este jogo deve ser modelado como um problema de procura com as seguintes definições:

- Estado - o pássaro e as caixas numa posição do mapa. Em atenção que as posições têm de estar dentro das regras acima referidas;
- Estado inicial - mapa inicial, onde o pássaro e as caixas ainda não se movimentaram;
- Estado final - todas as caixas já estão nas posições das estrelas;
- Natureza da solução pretendida - todas as operações que mudem o estado inicial para estado final. Neste caso, todos os movimentos que o pássaro faz colocar as caixas nas estrelas;
- Operadores de mudança de estado - todos os movimentos possíveis de realizar (Cima, Baixo, Esquerda, Direita);
- Custo associado a cada elemento - o custo associado é de 1 uma vez que o jogador só pode realizar um movimento de cada vez;
- Heurísticas - devem solucionar o problema com a solução mais económica.

Algoritmos de Procura

1. Procura Cega

Os algoritmos de procura cega limitam-se a encontrar soluções por geração de modo sistemático de novos estados que vão sendo comparados com o estado final pretendido.

1.1. Profundidade Limitada

O algoritmo de pesquisa em profundidade limitada é uma variante do algoritmo de profundidade primeiro. Como o de profundidade primeiro tem um problema com caminhos infinitos, neste limita-se o nível máximo da procura. Contudo este algoritmo tem um problema, saber o valor a utilizar como valor máximo para o limite.

Este algoritmo foi implementado no script “LimitedDepthSearch” partindo do script “BreathFirstSearch” fornecido pelos docentes. Neste processo foi necessário fazer duas alterações:

- Fazer uso de uma Stack em vez de Queue no “SearchNode”;
- Definir uma variável limite que é usada para verificar se a profundidade do nó atual é menor que o limite.

1.2. Aprofundamento Progressivo

Este algoritmo vai resolver o problema do algoritmo anterior. Para corrigir esse problema, utilizamos o algoritmo de profundidade limitada mas desta vez não definimos o valor máximo do limite, este vai sendo incrementado.

Este algoritmo está implementado no script “ProgressiveDepthSearch”. A única alteração efectuada foi na condição de quando a stack já não contém nós, aqui, o limite é incrementado, a stack e o hashset são limpos e volta-se a chamar a função Begin().

2. Procura Informada

Estes tipos de algoritmo utilizam conhecimentos sobre o problema na escolha do próximo nó a ser expandido com características do problema. Para a realização destes algoritmos tivemos de desenvolver heurísticas que serão explicadas mais à frente.

2.1. Pesquisa Sôfrega

O algoritmo de procura sôfrega consiste na escolha do nó mais promissor de acordo com o valor estimado por h(n). A árvore de procura é ordenada relativamente a h(n), sendo escolhido o nó com o valor mínimo que está mais próximo da solução pretendida.

O script “GreedySearch” implementa este algoritmo utilizando a heurística descrita no enunciado. Utiliza-se uma priority queue e uma variável que vai obter o valor devolvido pela heurística, “heu”. Desta forma, sempre que se expande um nó sucessor, a heurística é calculada com o estado deste sucessor. À priority queue vai ser adicionado esse nó e são todos ordenados por ordem crescente em função do valor de h.

Neste algoritmo é possível alterar a heurística que se pretende testar devido a uma variável pública criada para podermos escolher entre as heurística desenvolvidas.

2.2. A*

O algoritmo A* procura escolher a cada instante o melhor caminho passando pelo nó, utilizando a função f(n), que é definida por: f(n) = g(n) + h(n).

O algoritmo A* está implementado no script “AStarsearch” e é idêntica ao algoritmo anterior. A única diferença é a ordenação que é feita relativamente a f.

Tal como no algoritmo anterior, podemos alterar as heurísticas que se pretende utilizar através de uma variável pública.

Heuristicas

1. Goals Missing

A primeira heurística foi-nos fornecida pelos docentes: “a estimativa do custo de transitar de um estado s até ao estado final é igual ao número de caixas que não estão numa posição destino”. Para esta heurística utilizamos uma função GoalsMissing, na qual se obtém o número de objectivos e verifica se as caixas já está na posição destino. Se estiver, decrementa-se o valor de GoalsMissing. No final, retorna-o. Assim, para cada estado num determinado nó, verifica quantas caixas faltam colocar na posição objectivo.

2. Distancias

Os mapas de Sokoban podem ser vistos como um plano que contém elementos que se movem para determinadas posições. Deste modo, achámos que as melhores heurísticas a implementar deveriam estar relacionadas com a distância de uns elementos relativamente a outros.

Ambas as heurísticas são admissíveis, pois nenhuma delas calcula excessivamente a distância entre elementos, procurando sempre a distância mínima. Independentemente da distância euclidiana entre um objecto e outro ser menor do que a sua distância segundo a de Manhattan como o jogador não pode andar na diagonal, acaba por percorrer a mesma distância tanto numa heurística como na outra.

2.1. Distância de Manhattan

A fórmula $\sum|ai - bi|$, soma das diferenças absolutas das coordenadas de dois pontos, dá-nos a distância entre dois determinados pontos. Assim, implementámos duas funções distintas: a primeira soma as distâncias mínimas entre o jogador e as caixas (distCratePlayer); por sua vez, a segunda calcula as distâncias mínimas entre as caixas e as posições objectivo (distCrateGoal).

As funções que aplicam esta fórmula encontram-se no script “SokobanProblem” e podem ser utilizadas nos scripts “AStarsearch” e “GreedySearch”.

É de ter em conta que esta heurística deixaria de ser admissível se o jogador pudesse deslocar-se na diagonal.

2.2. Distância Euclidiana

Ao contrário da distância de Manhattan, a Euclidiana calcula a distância em linha recta entre dois pontos, através da fórmula $\sqrt{\sum}(ai - bi)2$. Novamente, implementámos duas funções com o cálculo das distâncias mínimas descritas em cima.

Novamente as funções relativas a esta fórmula encontram-se no script “SokobanProblem” e podem ser utilizadas nos scripts “AStarsearch” e “GreedySearch”.

Experimentação

Mapa 1

		Visited	Expanded	Actions
Breath First Search		22	58	5
Limited Depth Search		17	47	5
Progressive Depth Search		41	116	5
Pesquisa sôfrega	DistCratePlayer	9	27	5
	DistCrateGoal	21	55	5
	DistEuclGoal	21	55	5
	DistEuclPlayer	15	43	5
	DistGoalsMissing	21	55	5
A*	DistCratePlayer	6	18	5
	DistCrateGoal	19	50	5
	DistEuclGoal	19	50	5
	DistEuclPlayer	6	18	5
	DistGoalsMissing	19	50	5

Este mapa tem menor complexidade relativamente aos outros, pois tem apenas uma caixa, uma posição objectivo e não há obstáculos entre o jogador e a caixa. Sendo assim é normal que os nós visitados e expandidos sejam reduzidos. Dada a esta baixa dificuldade do mapa, em termos de complexidade temporal, obtivemos um valor muito elevado por aprofundamento progressivo em comparação com os restantes algoritmos. Com as heurísticas que calculam as distâncias entre a caixa e a posição objectivo e para a largura primeiro obtivemos resultados semelhantes.

Mapa 2

		Visited	Expanded	Actions
Breath First Search		386	1123	8
Limited Depth Search		66	201	10
Progressive Depth Search		767	2301	10
Pesquisa sôfrega	DistCratePlayer	97	296	8
	DistCrateGoal	46	145	10
	DistEuclGoal	64	200	100
	DistEuclPlayer	158	472	8
	DistGoalsMissing	187	564	8
A*	DistCratePlayer	63	194	8
	DistCrateGoal	36	115	10
	DistEuclGoal	36	115	10
	DistEuclPlayer	223	688	24
	DistGoalsMissing	30	96	8

Este mapa tem uma complexidade mais elevada que o anterior, pois apesar de não ter mais obstáculos entre o jogador e as caixas, tem mais caixas, ou seja, mais relações jogador-caixa, caixa-objectivo ou posições objectivo por atingir. Relativamente à complexidade temporal, tivemos novamente uma pior performance do aprofundamento progressivo, fazendo este o algoritmo menos eficiente.Quanto às heurísticas o que obteve o melhor resultado foi o fornecido previamente pelos docentes.

Mapa 3

		Visited	Expanded	Actions
Breath First Search		134251	382032	29
Limited Depth Search		2274	6260	35
Progressive Depth Search		105304	281586	35
Pesquisa sôfrega	DistCratePlayer	13918	38640	29
	DistCrateGoal	60807	19622	31
	DistEuclGoal	61079	170323	31
	DistEuclPlayer	18557	51873	29
	DistGoalsMissing	126745	362926	29
A*	DistCratePlayer	15730	44459	33
	DistCrateGoal	60467	168733	31
	DistEuclGoal	60467	168733	31
	DistEuclPlayer	10016	27950	43
	DistGoalsMissing	3506	9568	37

Ao contrário dos anteriores, o algoritmo de procura cega aprofundamento progressivo é mais eficiente que o de largura primeiro. Isto acontece pois a especificidade deste mapa é superior aos anteriores, ou seja, tem mais caixas e posições objectivo, sendo que, inicialmente, as próprias caixas são obstáculos para o jogador. Conseguimos concluir em termos de complexidade temporal, que todos os algoritmos de pesquisa são melhores que o largura primeiro e todos são melhores que o aprofundamento progressivo, sendo que o que obteve melhor performance temporal foi o profundidade primeiro.

Mapa 4

		Visited	Expanded	Actions
Breath First Search		13812	33596	50
Limited Depth Search		1176	2979	60
Progressive Depth Search		55443	141692	60
Pesquisa sôfrega	DistCratePlayer	7072	17407	50
	DistCrateGoal	5525	13583	50
	DistEuclGoal	7115	17446	50
	DistEuclPlayer	8732	21410	50
	DistGoalsMissing	13126	31822	50
A*	DistCratePlayer	6408	16067	74
	DistCrateGoal	5000	12419	66
	DistEuclGoal	4739	11782	66
	DistEuclPlayer	6006	15133	74
	DistGoalsMissing	9099	21967	52

O mapa 4 têm apenas 2 caixas mas como os caminhos requerem o reajustamento das caixas pois não é possível por 1 caixa de cada vez no objetivo pode se ver que têm uma complexidade maior logo requer um maior número de movimentos pelo que o LimitedDepthSearch obteve os melhores resultados visto já se saber qual o limite a dar .

Mapa 5

		Visited	Expanded	Actions
Breath First Search		708161	71617	35
Limited Depth Search		23586	67847	56
Progressive Depth Search		955331	2748378	56
Pesquisa sôfrega	DistCratePlayer	207555	595400	40
	DistCrateGoal	180509	497952	37
	DistEuclGoal	212478	587932	37
	DistEuclPlayer	255521	721957	38
	DistGoalsMissing	442260	1234389	38
A*	DistCratePlayer	272826	789320	47
	DistCrateGoal	81030	217410	52
	DistEuclGoal	45485	121937	52
	DistEuclPlayer	321651	915503	55
	DistGoalsMissing	1767	4641	39

Mapa 6

		Visited	Expanded	Actions
Breath First Search		71617	158845	89
Limited Depth Search		5275	11230	213
Progressive Depth Search		967207	2070370	213
Pesquisa sôfrega	DistCratePlayer	8911	19306	89
	DistCrateGoal	9808	21358	93
	DistEuclGoal	12315	26751	91
	DistEuclPlayer	9858	21520	89
	DistGoalsMissing	71123	157717	89
A*	DistCratePlayer	5795	12643	105
	DistCrateGoal	4768	10423	121
	DistEuclGoal	4898	10664	123
	DistEuclPlayer	5103	11182	105
	DistGoalsMissing	9542	21360	89

Mapa 8

		Visited	Expanded	Actions
Breath First Search		1073204	2802189	93
Limited Depth Search		149348	400668	1493
Progressive Depth Search		5866057	15681530	117
Pesquisa sôfrega	DistCratePlayer	263555	682344	93
	DistCrateGoal	280695	731135	99
	DistEuclGoal	343314	890434	99
	DistEuclPlayer	380974	1063578	93
	DistGoalsMissing	1063578	2775789	93
A*	DistCratePlayer	65593	175292	105
	DistCrateGoal	56444	144717	117
	DistEuclGoal	8020	21010	109
	DistEuclPlayer	97087	257490	105
	DistGoalsMissing	28793	73864	87

Algorítmos e complexidades

O aprofundamento progressivo vai analisar o mesmo nós várias vezes, fazendo com que haja uma sobrecarga temporal, dando uma complexidade elevada que se pode verificar nos resultados obtidos em quase todos os mapas, dado que este algoritmo tinha sempre o tempo maior. Conseguimos ver ainda que, dado que o espaço mínimo depende da profundidade da solução, nos mapas com uma maior variedade de caminhos possíveis os seus valores são tendencialmente maiores. Em relação à profundidade limitada, dado termos definido um limite, o algoritmo sabe que há um máximo ao qual pode chegar, nunca correndo o risco de chegar a um ponto infinito de pesquisa. Como tal, a pesquisa que este algoritmo vai executar com tempos bastante mais reduzidos em comparação aos outros algoritmos, o que também se verificou com os nossos resultados. A pesquisa sôfrega e o A* têm a mesma complexidade, mas o primeiro algoritmo como apenas avalia o próximo caminho com menos custo e não o custo mais pequeno na sua totalidade, acaba por ter tempos inferiores mas com caminhos mais caros, o que se pode verificar ao comparar os nossos resultados.

		Completo	Discriminador	Tempo Mínimo	Espaço Mínimo
Procura Cega	Breadth First	Sim	Sim	O(b^p)	O(b^p)
	Depth Limited	Sim	Não	O(b^l)	O(b*l)
	Progressive Depth	Sim	Não	O(b^p)	O(b*p)
Procura Informada	A*	Sim	Sim	O(b^p)	O(b^p)
	Greedy	Sim	Não	O(b^p)	O(b^p)

b = factor de ramificação (no nosso caso, 4)
p = profundidade da solução
l = limite definido

Conclusão

Com a realização do segundo trabalho prático, conseguimos consolidar os conhecimentos gerais sobre agentes de procura e seus algoritmos. Relativamente, à parte da modelação não encontramos dificuldades em definir o Sokoban como um problema de pesquisa. A implementação dos diferentes algoritmos de procura já foi um pouco mais complicada, pois tivemos alguns erros a nível de código que só foi possível descobrir aquando da experimentação, pois estávamos a obter resultados
incomuns.