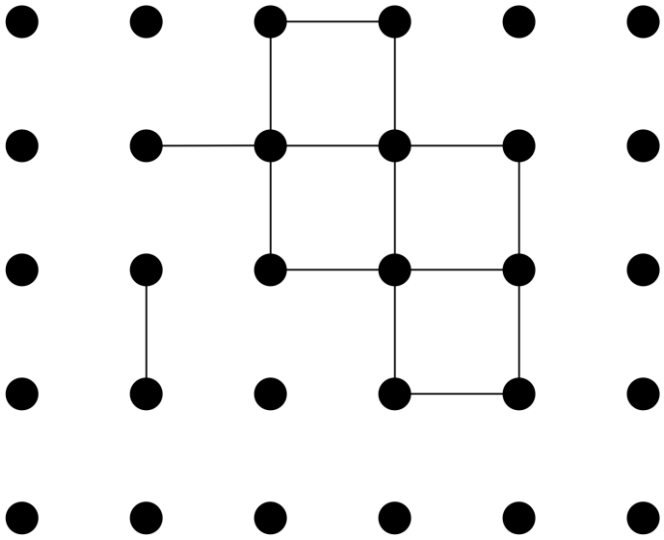
Manual Técnico



Puzzle dos Pontos e das Caixas

Docentes da disciplina

Joaquim Filipe

Cédric Grueau

Um trabalho realizado por

Artur Esteves 140221076 Engenharia Informática

Daniel Costa 120221058 Engenharia Informática

Conteúdo

Introdução

No âmbito da disciplina Inteligência Artificial, do curso de Engenharia informática do Instituto Politécnico de Setúbal – Escola Superior de Tecnologia de Setúbal, foi proposto pelos docentes da disciplina o desenvolvimento de um projeto, em LISP, denominado por "Puzzle dos pontos e caixas", de modo que os alunos possam aplicar os seus conhecimentos adquiridos nas aulas práticas e laboratórios da disciplina.

Este jogo consiste em fechar um determinado número de caixas a partir de uma configuração inicial do tabuleiro. Para alcançar o objetivo será necessário desenhar um arco entre dois pontos adjacentes, tanto na horizontal como vertical, e quando o tiver 4 arcos desenhados, entre pontos adjacentes, temos uma caixa fechada.

Para mais informações sobre as características do jogo é favor consultar o enunciado do projeto de forma a integrar-se no jogo e o seu modo de procedimento.

Ambiente de desenvolvimento

Software

A Tabela 1 Softwares utilizados, contém os softwares utilizados para o desenvolvimento deste projeto:

TABELA 1 SOFTWARES UTILIZADOS

LispWorks 7.0 (x64)	Ferramenta de desenvolvimento
Notepad++	Editor de Texto
GIT	Repositório

Requisitos

A Tabela 2 Requisitos mínimos software, contém os requisitos mínimos e recomendados das máquinas de modo a suportar o LispWorks 7.0 (64bits):

TABELA 2 REQUISITOS MÍNIMOS SOFTWARE

Requisitos mínimos	
Windows (x64)	Windows (x32)
Processador AMD64|Intel 64 Memória de disco: 170MB Sistema Operativo: Windows Vista SP2 (x64)	Processador Pentium 4 Memória de disco: 170MB Sistema Operativo: Windows Vista SP2 (x64)
Macintosh (x64)	**Macintosh (x32)**
Processador Intel-based Macintosh Memória de disco: 285MB Sistema Operativo: OS X v.10.5	Processador Intel-based Macintosh Memória de disco: 265MB Sistema Operativo: OS X v.10.5

Algoritmo geral

Conteúdo da aplicação

Este projeto foi proposto com a seguinte estrutura:

- Problemas.dat

  Ficheiro que contêm os tabuleiros, isto é, os problemas que serão para resolver pelo programa. Cada problema é uma lista que contém uma lista de arcos verticais e arcos horizontais.

- Projeto.lisp

  Ficheiro responsável pela interação com o utilizador, imprimindo menus no ecrã da máquina com as opções.

- Puzzle.lisp

  Ficheiro responsável pela gestão das regras do domínio da aplicação. Neste ficheiro são encontrados os operadores, validações e outras funções úteis para o seu desenvolvimento.

- Procura.lisp

  Neste ficheiro encontra-se a procura genérica e os algoritmos, assim como os sucessores, funções de cálculo (penetrância e fator de ramificação) e funções auxiliares. Responsável pela execução da procura de um problema até encontrar a solução objetiva.

NOTA:
Gostaríamos de alertar para a utilização dos ficheiros nos diversos Sistemas Operativos (Mac OS X; LINUX; Elementary entre outros), na qual não tínhamos qualquer meio de utilizar nesses mesmos ambientes.

Algoritmos

Neste projeto foi proposto 4 algoritmos, na qual ao longo das próximas secções vamos descrever algumas das suas características.

Breadth-First

A procura em largura (em Inglês Breadth-first) é um método de procura não-informada que expande todos os nós de uma árvore de uma solução. Este método realiza uma procura exaustiva numa árvore inteira, sem considerar o seu alvo de procura, até que ele o encontre. Este algoritmo não utiliza uma heurística.
Do ponto de vista do algoritmo, todos os nós filhos obtidos pela expansão de um nó são adicionados a uma fila (FIFO). Normalmente, nós que ainda não foram estudados pelos seus vizinhos são colocados num *container* (uma lista) designada por “lista de aberto". Uma vez examinados, são colocados num outro *container, a “lista de fechados”.*

Depth-First

A procura em profundidade (*Depth-first* em Inglês) é um algoritmo de procura em profundidade realiza uma procura, não-informada, que progride através da expansão do primeiro nó filho da árvore de procura, e se aprofunda cada vez mais, até que o alvo da procura seja encontrado ou até que ele se depare com um nó que não possui filhos (nó folha). Então a procura retrocede (*backtrack*) e começa no próximo nó.

A*

A procura A* é um tipo de procura informada (método heurística), ao contrário dos algoritmos já apresentados, que utiliza heurísticas de forma a direcionar a procura de uma solução. Recorrendo a uma heurística e estimando o custo de cada nó até a solução (composto por **g** (custo até ao nó em questão) e **h** (custo restante até à solução) é possível ordenar os nós a serem expandidos e assim explorar os que são mais prováveis de levar a uma solução. Desta forma reduz-se o número de nós a expandir tornando a pesquisa mais rápida.

IDA*

O IDA* utiliza o mesmo conceito que o A*, mas só que tenta reduzir a quantidade de memória utilizada através do uso de limiares de custo, **L.**
Isto é só mantém em memória, e expandem-se, nós que tenham um valor inferior ao limite estimado no nó inicial. Caso se chegue a um caso em que não se encontram nós que satisfaçam esta condição, volta a fazer executar a procura e escolhe um novo limiar, que assumo como limite o valor mais baixo por entre os nós descartados. Repete-se este procedimento até encontrar a solução.

NOTA:
Tanto o algoritmo A* como IDA* algoritmos encontram a solução ótima caso a heurística utilizada seja admissível.

Heurística

As heurísticas acrescentam conhecimento sobre o domínio da aplicação aos algoritmos de procura informados (A* e IDA*). São regras empíricas que podem oferecer aceleração no processo de procura. As heurísticas são admissíveis quando, para qualquer grafo, descobre sempre o caminho ótimo até achar o objetivo. Neste projeto foi desenvolvido duas heurísticas, sendo uma proposta pelos docentes e outra ao nosso critério.

- Heuristica1

  Fornecida pelo enunciado do projeto, consiste numa subtração do número de caixas fechadas de um tabuleiro pelo número de caixas ainda a fechar, escritas pelo utilizador e ainda subtrai 1. Esta heurística é admissível, visto que o valor heurístico da solução dá 0 (h=0) (Ver Resultados). A forma apresentada será a seguinte:

  - **h(x) = o(x) + c(x) – 1**, onde o(x) é o numero de caixas a fechar e c(x) o número de caixas fechadas de um determinado tabuleiro.

- Heuristica2

  Heurística é proposta pelos alunos que realizaram o projeto onde consiste numa subtração do número de caixas fechadas de um tabuleiro pelo número de caixas ainda a fechar. Trata-se de uma heurística simples e eficaz e para alguns casos é admissível. A forma apresentada será a seguinte:

  - **h(x) = o(x) + c(x),** onde o(x) é o numero de caixas a fechar e c(x) o número de caixas fechadas de um determinado tabuleiro.

Funções

Segue uma lista das funções principais, existentes em cada ficheiro, assim como a sua descrição:

- Projeto.lisp

Função	Descrição
(iniciar)	Função que inicializa o programa, chama a função que pede ao utilizador colocar a diretoria e esse será passada como argumento para carregar os ficheiros.
(inserir-diretoria)	Função que pede ao utilizador para colocar a raiz da pasta onde se encontram os ficheiros do projeto.
(load-files)	Função que carrega os ficheiros de puzzle e procura para o programa e executa o menu"
(menu-inicial)	Apresenta o menu principal do programa.
(iniciar-procura)	Função que pede ao utilizador toda a informação necessária para começar uma procura no espaço de estados.
(ler-tabuleiro)	Lista todos os problemas e faz a leitura numa diretoria que chama o ficheiro que contém todos os problemas.
(ler-numero-objectivo-caixas)	Lê o numero de caixas que o utilizador pretende fechar.
(ler-algoritmo)	Função que recebe do utilizador o algoritmo que pretende utilizar do problema anteriormente escolhido.
(ler-heuristica)	No caso de o utilizador escolher o algoritmo a-asterisco ou ida-asterisco, será mostrado o menu que mostra as duas heurísticas que pretende usar na procura
(ler-profundidade)	Função que lê a profundidade máxima do utilizador caso esta escolha o algoritmo depth-first (procura em profundidade).
(resultados)	Imprime os resultados do jogo.
(sem-resultados)	Função que imprime num ficheiro do tipo .DAT que não existe solução de determinado nó.

- Procura.lisp

Função	Descrição
(procura-generica)	Permite procurar a solução de um problema usando a procura no espaço de estados. A partir de um estado inicial, de uma função que gera os sucessores e de um dado algoritmo
(procura-ida-asterisco)	Permite procurar a solução de um problema usando procura no espaço de estados. A partir de um estado inicial, gera os sucessores e do algoritmo IDA* a partir de um estado inicial e um limiar.
(novo-limiar)	Função auxiliar da procura do algoritmo IDA*, que serve para implementar um novo limiar caso seja maior que o custo.

(bfs)	Função que executa o algoritmo de procura em largura. Faz uma junção da lista de abertos com os sucessores.
(dfs)	Função que executa o algoritmo de procura em profundidade. Faz uma junção da lista de sucessores com os abertos.
(a-asterisco)	Função que executa o algoritmo a*. Faz uma junção da lista de sucessores com os abertos, tendo em conta a ordenação do valor do custo.
(ida-asterisco)	Função que executa o algoritmo ida*. Faz uma validação do custo do nó com o limiar que é passado pelo argumento, caso contrário passará ao próximo nó sucessor.
(sucessores)	Função que retornada uma lista com todos os sucessores de um nó.
(existep)	Retorna valor verdadeiro, se o nó existir na lista. Para o algoritmo dfs, o conceito de nó repetido é particular.
(existe-solucao)	Verifica se existe uma solução ao problema numa lista de sucessores para o algoritmo dfs
(penetrancia)	Retorna o valor da penetrância dos nos gerados até o nó objetivo sobre dos nos totais gerados.
(fator-ramificacao)	Retorna o valor do fator de ramificação, compreendido entre uma margem.

- Puzzle.lisp

Função	**Descrição**
(criar-no)	Cria uma lista que representa um nó. Um nó é composto pelo estado que é o tabuleiro, este é um parâmetro obrigatório, é composto também por outros parâmetros, como a profundidade a que se encontra, pela heurística deste mesmo nó e pelo nó pai, ou seja, o nó que o gerou. A profundidade e a heurística por omissão têm valor nil, enquanto que o nó pai por defeito é NIL.
(get-no-estado)	Retorna o estado do nó, que é representado pelo tabuleiro.
(get-no-profundidade)	Retorna a profundidade em que o nó se encontra.
(get-no-heuristica)	Retorna a heurística do nó.
(get-no-pai)	Retorna o nó pai deste nó, ou seja, o nó que gerou este nó.
(custo)	Retorna o valor do custo do nó (f). Soma do valor da profundidade com o valor heurístico.
(get-arcos-horizontais)	Retorna a lista dos arcos horizontais de um tabuleiro.
(get-arcos-verticais)	Retorna a lista dos arcos verticais de um tabuleiro.
(Inserir-arco-na-posicao)	Insere um arco (representado pelo valor [T]) numa lista que representa o conjunto de arcos dum tabuleiro. A posição representada pela linha e a coluna de destino são valores inteiros passados como argumentos.

(inserir-arco-vertical)	Insere um arco vertical (representado pelo valor [T]) num tabuleiro passado como argumento.
(inserir-arco-horizontal)	Insere um arco horizontal (representado pelo valor [T]) num tabuleiro passado como argumento.
(possível-adicionar-arco)	Recebe índices de linha e coluna e uma lista de arcos horizontais ou de arcos verticais e verifica se naquela posição o valor é [T], se for devolve [NIL], se for [NIL] devolve [T].
(operadores)	Cria uma lista com todos os operadores do problema dos Pontos e das Caixas.
(numero-linhas-tabuleiro)	Retorna o número de linhas verticais de um tabuleiro.
(numero-colunas-tabuleiro)	Retorna o número de linhas verticais de um tabuleiro.
(caixas-fechadas)	Retorna o numero de caixas fechadas de um tabuleiro.
(contar-objetivo)	Função que irá contar se a caixa está fechada ou não, isto é, se a função auxiliar conta-caixa-fechada
(contar-nils-lista)	Função que irá contar o numero de NILS's existentes numa lista. Se não existir dará valor 0, ou seja, teríamos uma lista só com T, o que resulta uma caixa fechada.
(heuristica1)	Usada uma heurística que privilegia os tabuleiros com o maior número de caixas fechadas.
(heuristica2)	Heurística efetuada pelos alunos que se baseia na heuristica1.
(solucaop)	Devolve [T] se o número de caixas a fechar for igual ao número de caixas fechadas do nó, e devolve [NIL] se não for
(caminho-solucao)	Retorna o caminho até à solução. Ou seja, todos os estados desde a inicial à solução.

Diagrama - Interação com Utilizador

O diagrama de sequência que se segue, é uma representação de baixo nível, da interação que o utilizador terá quando inicia o programa na consola.

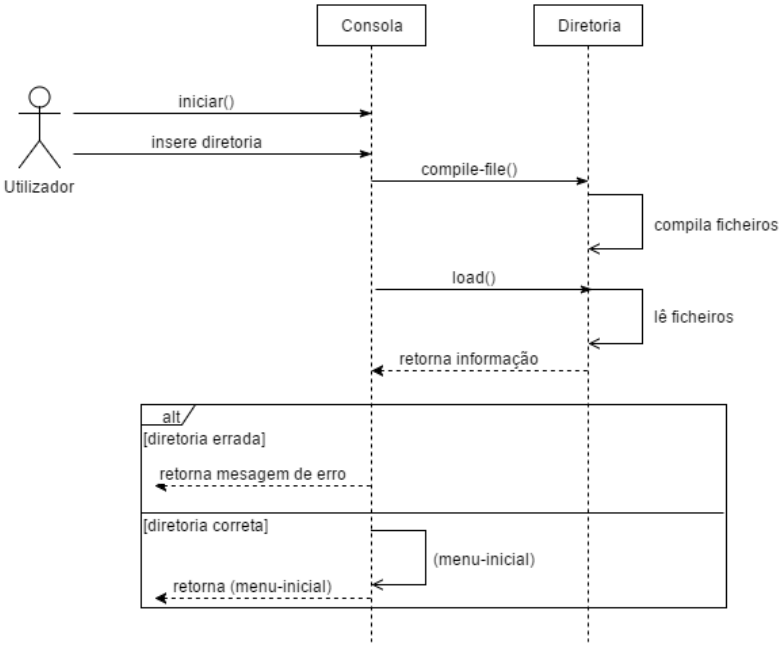


ILUSTRAÇÃO 1 DIAGRAMA SEQUÊNCIAS DA INTERAÇÃO INICIAL

Análise dos resultados

Resultados

Os resultados seguintes, serão mostrados por ordem dos problemas, tento em conta o número de caixas objetivas apresentadas pelo docente:

Problema A para 3 caixas objetivo

	BFS	DFS	A* - Heuristica1	A* - Heuristica2	IDA* - Heuristica1	IDA* - Heuristica2
Nós Gerados	1129	84	313	1129	127	127
Nós Expandidos	81	31	22	81	13	13
Profundidade	2	4	2	2	12	12
Penetrância	0.0017714792	0.04761905	0.0063897763	0.0017714792	0.09448819	0.09448819
Fator de ramificação	8.50333	2.1368685	4.410605	8.50333	1.2273736	1.2273736
Caixas Fechadas	3	3	3	3	3	3
Valor Heurístico	NIL	NIL	0	2	0	5/2
Tempo decorrido(s)	0	0	0	0	0	0

Problema B com 7 caixas objetivo

	BFS	DFS	A* - Heuristica1	A* - Heuristica2	IDA* - Heuristica1	IDA* - Heuristica2
Nós Gerados	198	16	198	198	85	85
Nós Expandidos	15	1	15	15	8	8
Profundidade	1	1	1	1	7	7
Penetrância	0.005050505	0.0625	0.005050505	0.005050505	0.08235294	0.08235294
Fator de ramificação	15.324541	1.682121	15.324541	15.324541	1.2273736	1.2273736
Caixas Fechadas	7	7	7	7	7	7
Valor Heurístico	NIL	NIL	1	2	0	13/2
Tempo decorrido(s)	0	0	0	0	0	0

Problema C com 10 caxas objetivo

	BFS	DFS	A* - Heuristica1	A* - Heuristica2	IDA* - Heuristica1	IDA* - Heuristica2
Nós Gerados	-	6400	76968	-	162	162
Nós Expandidos	-	6311	6710	-	15	15
Profundidade	-	10	8	-	14	14
Penetrância	-	0.0015625	1.039393E-4	-	0.086419754	0.086419754
Fator de ramificação	-	2.1368685	3.0463629	-	1.2273736	1.2273736
Caixas Fechadas	-	10	10	-	10	10
Valor Heurístico	-	Nil	Nil	-	0	19/2
Tempo decorrido(s)	-	2	232	-	0	0

Problema D com 10 caixas objetivo

	BFS	DFS	A* - Heuristica1	A* - Heuristica2	IDA* - Heuristica1	IDA* - Heuristica2
Nós Gerados	-	1171	-	-	1171	1171
Nós Expandidos	-	39	-	-	40	40
Profundidade	-	39	-	-	39	39
Penetrância	-	0.033304867	-	-	0.033304867	0.033304867
Fator de ramificação	-	1.2273736	-	-	1.2273736	1.2273736
Caixas Fechadas	-	10	-	-	10	10
Valor Heurístico	-	Nil	-	-	0	19/2
Tempo decorrido(s)	-	0	-	-	0	0

Problema E com 20 caixas objetivo

	BFS	DFS	A* - Heuristica1	A* - Heuristica2	IDA* - Heuristica1	IDA* - Heuristica2
Nós Gerados	-	796	-	-	796	796
Nós Expandidos	-	30	-	-	31	31
Profundidade	-	30	-	-	30	30
Penetrância	-	0.03768844	-	-	0.03768844	0.03768844
Fator de ramificação	-	1.2273736	-	-	1.2273736	1.2273736
Caixas Fechadas	-	20	-	-	20	20
Valor Heurístico	-	NIL	-	-	1	19
Tempo decorrido(s)	-	0	-	-	0	0

Problema F com 45 caixas objetivo

	BFS	DFS	A* - Heuristica1	A* - Heuristica2	IDA* - Heuristica1	IDA* - Heuristica2
Nós Gerados	-	5986	-	-	5986	5986
Nós Expandidos	-	105	-	-	106	106
Profundidade	-	105	-	-	105	105
Penetrância	-	0.017540928	-	-	0.017540928	0.017540928
Fator de ramificação	-	1.2273736	-	-	1.2273736	1.2273736
Caixas Fechadas	-	45	-	-	45	45
Valor Heurístico	-	Nil	-	-	0	89/2
Tempo decorrido(s)	-	1	-	-	0	0

NOTA:
Os problemas C, D, E e F, não estão preenchidos, em alguns algoritmos, uma vez que apresentam erros de memória (*You are approaching the heap size limit for the Personal Edition of LispWorks.If you choose to continue now you are advised to save your work at regular intervals*.).
Ou seja, o programa LispWorks está intencionalmente limitado,visto que é uma versão gratuita, de modo a evitar os chamados "crashes" com outros programas. Assim, este problema acaba por ser uma limitação que apresentamos (Limitações da aplicação).

Quadro Comparativo

No quadro seguinte (Tabela 3 Quadro comparativo dos resultados), podemos ter uma ideia do tempo de execução (em segundos) para cada problema, utilizando algoritmos diferentes:

TABELA 3 QUADRO COMPARATIVO DOS RESULTADOS – TEMPO DE EXECUÇÃO

Problemas	Breadh-First	Depth-First	A* (heuristica1)	A* (heuristica2)	IDA* (heuristica1)	IDA* (heuristica2)
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	-	2	232	-	0	0
D	-	0	-	-	0	0
E	-	0	-	-	0	0
F	-	1	-	-	0	0

Este quadro apresenta os tempos de execução dos exemplos acima mostrados na Tabela 3 Quadro comparativo dos resultados – Tempo de execução.

Limitações da aplicação

Neste projeto existem algumas limitações. Deparamos com uma limitação do LispWorks, que para resolver alguns problemas, apresenta problemas de "*heap size*", este problema deve-se ao facto de o LispWorks na versão gratuita disponibilizar apenas uma pouca quantidade de memória e como alguns os problemas que resolvemos já começam a ser pesados devido às dimensões das árvores geradas. Os algoritmos não informados, como a procura em largura e a profundidade encontram a solução ótima se os custos forem uniformes, no entanto devido à quantidade de nós que a profundidade em largura tem de analisar muitas das vezes não é possível encontrar a solução com este algoritmo. Algoritmos informados como o A* e IDA* só encontram a solução ótima se a heurística utilizada for admissível, no entanto como a função que gera os sucessores não está otimizada irá atrasar a procura.

Conclusão

Com o desenvolvimento deste primeiro projeto, ao longo das últimas semanas foi bastante enriquecedor no que toca aos assuntos abordados de LISP, e são sem dúvida uma mais valia na futura vida académica e profissional, no entanto, e de notar que é uma linguagem que apresenta um grau de dificuldade exigente. Inicialmente, encontramos algumas dificuldades de codificação, mas assim que essa foi ultrapassada deparamo-nos com erros, de limitação de memória, ao testar a solução dos problemas, que acabou por ser uma limitação deste projeto. Em relação aos algoritmos foram todos bem-sucedidos, e de modo eficaz, isto é, o programa está capaz de apresentar a solução do problema num curto espaço de tempo, no entanto, podemos concluir que na procura dos algoritmos não informados, o Breadth-First é o que gera maior número de nós, comparativamente ao Depth-First, o que leva a ocupar mais memória, sendo menos eficiente. Relativamente aos algoritmos informados, devido ao uso da heurística, podemos verificar que geram menor número de nós comparativamente ao Depth-First e Breadth-First, o que leva a ocupar menor memória do programa.

Bibliografia

Fontes Literárias

- Filipe, Joaquim (2015) - Algoritmos de Procura em Espaço de Estados, Setúbal.

Fontes de endereço

- LispWorks. Disponível em http://www.lispworks.com/support/system-requirements.html . Acesso em 7 de dezembro de 2016.
- Design and Analysis of Algorithms. Disponível em https://www.ics.uci.edu/~eppstein/161/960215.html. Acesso em 2 de Dezembro de 2016.
- University of Washington, Computer Science & Engineering. Disponível em http://faculty.washington.edu/dbp/SAPACLISP-1.x/basic-math.lisp. Acesso em 23 de novembro de 2016.
- University of Washington, Computer Science & Engineering. Disponível em https://courses.cs.washington.edu/courses/cse415/06wi/notes/IDA.pdf. Acesso em 29 novembro de 2016.
- Heap Size, LispWorks User Guide - 11 Mar 2008. Disponível em http://www.lispworks.com/documentation/lw51/LWUG/html/lwuser-456.htm . Acesso em 17 dezembro de 2016.
- Prof. Michel Gagnon, Resolução de Problemas. Disponível em http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/IA/ResolProb/resproblema.html. Acesso em 18 dezembro de 2016.