Manual Técnico

Inteligência Artificial - Escola Superior de Tecnologia de Setúbal

2024/2025

Prof. Joaquim Filipe

Eng. Filipe Mariano

202000634 Bruno Ascenção

202000584 Francisco Pereira

Projeto Nº 1: Época Normal

20/12/2024

Índice

1. Arquitetura do sistema

O projeto contém 4 módulos, estão abaixo descritos quais são e para que servem:

1.1. puzzle.lisp

Este módulo contém tudo o que está relacionado com o problema como por exemplo, os operadores, getters e setters.

1.2. search.lisp

Este módulo contém toda a lógica dos algoritmos, BFS, DFS e A*.

1.3. branching.lisp

Este módulo contém a lógica relacionada com o cálculo da bissecção.

1.4. project.lisp

Este módulo é o cérebro do projeto. Utiliza todos os outros módulos para resolver o problema.

1.5. Conexões e relacionamentos

Como referido anteriormente o project.lisp faz a gestão de todos os módulos atuando como o cérebro do programa. Isso significa que não existe uma ligação entre os módulos, existe sim uma ligação geral de todos os módulos com o project.lisp, sendo assim este módulo desempenha o papel de mediador entre os restantes.

Seguindo a lógica de execução do programa, são realizadas as seguintes trocas de informação:

- project utiliza o puzzle para criar nó inicial consoante o problema escolhido pelo utilizador
- project envia para o search o nó, as dependências e qual o algoritmo a ser executado
- search executa o algoritmo e devolve ao project os resultados obtidos
- project executa o branching para o cálculo da ramificação média, utilizando os dados recebidos do search
- branching devolve o cálculo da ramificação média ao project

2. Entidades e sua implementação

2.1. Nós

Os nós estão estruturados através de uma lista com quatro elementos:

- Estado;
- Profundidade;
- Valor heurístico;
- Nó pai.

```
(((7 1 7 7 7 0) (1 16 15 1 11 2)) 28 0 ((6 0 6 6 6 8) (0 15 14 1 11 2)) 27 0 nil)
```

3. Algoritmos e sua implementação

Neste projeto foram implementados três algoritmos que são os seguintes

3.1. BFS

O bfs, breadth first search, é um algoritmo de pesquisa em largura. Isto é, apenas avança para o nível seguinte após gerar todos os sucessores do nível atual. Enquanto são gerados os novos nós, o bfs vai garantir que não irá gerar nós já existentes, evitando assim loops infinitos.

A validação do nó solução é feita na geração dos sucessores, ou seja, quando são criados os novos nós é realizada ao mesmo tempo a validação se algum desses nós é a solução do problema.

```
(DEFUN bf(dependencies opened-list &optional (closed-list NIL))
  "Method to execute the breath-first algorithm"
  (IF (NULL opened-list) ;Error if the open list is empty
    "Failure on search"
    (PROGN
      (format-node (CAR opened-list)) ;Print to see if algorithhm is working
      (LET*
        (
          (valid-successors (spawn-successors-not-closed-not-opened dependencies (CAR
opened-list) opened-list closed-list)) ;Generate all successors
          (solution (solution-from-successors dependencies valid-successors)) ;Get the
solution if there is
        )
        (IF solution ;Verify if solution
          ;Return the solution and the algorithm data, append the successors to the
end of the rest of the opened list, add the node to the closed list
          (return-data solution (APPEND (CDR opened-list) valid-successors) (CONS (CAR
opened-list) closed-list)) ;Return solution and the data
          (bf dependencies (APPEND (CDR opened-list) valid-successors) (CONS (CAR
opened-list) closed-list));Go to next iteration
        )
      )
    )
  )
)
```

3.2. DFS

O dfs, depth first search, contrariamente ao bfs é um algoritmo de procura em profundidade. Isto é, o algoritmo gera sempre primeiro os sucessores do primeiro sucessor do nó atual. Por sem um algoritmo em profundidade, é necessário indicar o nível máximo a atingir com o objetivo de evitar a geração contínua de nós sucessores.

De forma igual ao bfs, neste algoritmo é necessário garantir que não é permitido gerar nós já existentes, para que mais uma vez, não existam loops infinitos.

Também como o bfs, se algum nó sucessor gerado for a solução, o mesmo será retornado.

```
(DEFUN df(dependencies max-depth opened-list &optional (closed-list NIL))
  "Method to execute the depth-first algorithm"
  (IF (NULL opened-list) ;Error if the open list is empty
    "Failure on search"
    (PROGN
      (format-node (CAR opened-list)) ;Print to see if algorithhm is working
      ;Validate the current node depth
      (IF (>= (FUNCALL (dependency-depth dependencies) (CAR opened-list)) max-depth)
;Current node depth is higher than the max allowed depth
        (df dependencies max-depth (CDR opened-list) (CONS (CAR opened-list) closed-
list)) ;Go to next iteration without spawning successors
        ;Max depth not reach so keep executing
        (LET*
          (
            (valid-successors (spawn-successors-not-closed-not-opened dependencies
(CAR opened-list) opened-list closed-list)) ;Generate all successors
            (solution (solution-from-successors dependencies valid-successors)) ;Get
the solution if there is
          )
          (IF solution ;Verify if solution
            ;Return the solution and the algorithm data, append the successors to the
end of the rest of the opened list, add the node to the closed list
            (return-data solution (APPEND valid-successors (CDR opened-list)) (CONS
(CAR opened-list) closed-list)) ;Return solution and the data
            (df dependencies max-depth (APPEND valid-successors (CDR opened-list))
(CONS (CAR opened-list) closed-list));Go to next iteration
          )
        )
      )
    )
  )
)
```

3.3. A*

O A*, contrariamente aos algoritmos já antes falados, avalia os nós utilizando uma heurística. Esta heurística serve para que o algoritmo ao realizar a pesquisa, ordene os sucessores de forma crescente consoante o valor heurístico. Isto permite que o A* chegue de forma mais rápida ao nó solução devido à ordenação dos sucessores, fazendo assim com que ao tentar expandir um nó e este tenha o valor heurístico 0, está subentendido que este será o nó solução.

Igualmente aos outros dois algoritmos já falados, este também garante que não será gerado nenhum nó sucessor já existente, evitando os loops infinitos.

```
(DEFUN a*(dependencies heuristic opened-list &optional (closed-list NIL))
  "Method to execute the a* algorithm, recieving the puzzle dependencies the
heuristicm method and the first node as a list on the opened list"
  (IF (NULL opened-list) ;Error if the open list is empty
    "Failure on search"
    (PROGN
      (format-node (CAR opened-list)) ;Print working node
      ;Validate if current node is the solution
      (IF (FUNCALL (dependency-is-solution dependencies) (CAR opened-list)) ;Verify if
```

```
node to be worked is solution
        (return-data (CAR opened-list) opened-list closed-list) ;Node is the solution
return data~
        ;Node not solution so keep executing
        (LET
          (
            (valid-successors (spawn-successors-not-closed-not-opened dependencies
(CAR opened-list) opened-list closed-list heuristic)) ;Generate all successors
          )
          (a* dependencies heuristic (a*_add-to-open dependencies valid-successors
(CDR opened-list)) (CONS (CAR opened-list) closed-list)) ;Go to next iteration
        )
      )
    )
  )
)
```

4. Descrição das opções tomadas

4.1. Dependências

As dependências é uma lista de métodos pertencentes ao módulo do puzzle, estes métodos simulam métodos dentro dos obejtos na progamação orientada a objetos. Isto permite que os algoritmos tenham acesso indireto aos getters e setters dos nós, sem estarem diretamente ligados com o módulo puzzle.

5. Limitações técnicas e desenvolvimento futuro

5.1. Limitações técnicas

- DFS - Não existe o recálculo da profundidade, no caso de haver um sucessor já fechado mas tem menor profundidade;
- A* - Não existe o recálculo da heurística quando é encontrado um sucessor já fechado mas com menor valor heurístico;
- Algoritmos extras não foram realizados;
- Acreditamos que a heurística desenvolvida não seja a mais eficiente, contudo é funcional;
- Devido ao funcionamento do BFS e DFS, muitos dos problemas dados no enunciado do projeto não puderam ser realizados já que o LispWorks é limitado na memória e em alguns problemas essa memória esgota-se antes do algoritmo terminar a resolução do problema.

5.2. Desenvolvimento futuro

- Algoritmos extras pedidos no enunciado;
- Refactoring do código para melhor análise;
- Utilização de closures para um uso mais eficiente da memória e assim possibilitar a que os algoritmos BFS e DFS possam talvez finalizar os problemas que agora não conseguem;

- Corrigir os problemas mencionados nas limitações técnicas.

6. Testes e resultados

Todos os dados inseridos estão guardados na pasta "statistics" em documentos de texto. Estes são gerados pelo programa no final da execução do algoritmo.

ABF - Average Branching Factor

6.1. BFS

Problema	Nós gerados	Nós expandidos	g(x) Profundidade	Penetrância	ABF	Tempo(s)
((0 0 0 0 0 2) (0 0 0 0 4 0))	22	10	4	0.18181819	1.8208008	0.0
((2 2 2 2 2 2) (2 2 2 2 2 2))	Na	Na	Na	Na	Na	Na
((0 3 0 3 0 3) (3 0 3 0 3 0))	2267	626	6	0.0026466695	3.415928	0.142
((1 2 3 4 5 6) (6 5 4 3 2 1))	Na	Na	Na	Na	Na	Na
((2 4 6 8 10 12) (12 10 8 6 4 2))	Na	Na	Na	Na	Na	Na
((48 0 0 0 0 0) (0 0 0 0 0 48))	Na	Na	Na	Na	Na	Na
((8 8 8 8 8 8) (8 8 8 8 8 8))	Na	Na	Na	Na	Na	Na

6.2. A*

Problema	Heurística	Nós gerados	Nós expandidos	g(x) Profundidade	Penetrância	ABF	Tempo(s)
((0 0 0 0 0 2) (0 0 0 0 4 0))	Default	12	5	6	0.5	1.2011719	0.001
((0 0 0 0 0 2) (0 0 0 0 4 0))	Custom	12	5	6	0.5	1.2011719	0.001
((2 2 2 2 2 2) (2 2 2 2	Default	78	13	14	0.17948719	1.2092285	0.003

2 2))							
((2 2 2 2 2 2) (2 2 2 2 2 2))	Custom	69	12	12	0.17391305	1.2550049	0.003
((0 3 0 3 0 3) (3 0 3 0 3 0))	Default	39	9	10	0.25641027	1.2473145	0.002
((0 3 0 3 0 3) (3 0 3 0 3 0))	Custom	35	9	9	0.25714287	1.2731934	0.002
((1 2 3 4 5 6) (6 5 4 3 2 1))	Default	274	42	35	0.12773723	1.095398	0.022
((1 2 3 4 5 6) (6 5 4 3 2 1))	Custom	359	93	29	0.08077995	1.1448822	0.028
((2 4 6 8 10 12) (12 10 8 6 4 2))	Default	282	40	41	0.14539007	1.0757446	0.013
((2 4 6 8 10 12) (12 10 8 6 4 2))	Custom	704	238	46	0.06534091	1.090332	0.085
((48 0 0 0 0 0) (0 0 0 0 0 48))	Default	616	163	45	0.073051945	1.0997315	0.125
((48 0 0 0 0 0) (0 0 0 0 0 48))	Custom	453	117	46	0.10154525	1.0852203	0.035
((8 8 8 8 8 8) (8 8 8 8 8 8))	Default	336	47	46	0.13690476	1.0715332	0.017
((8 8 8 8 8 8) (8 8 8 8 8 8))	Custom	422	86	49	0.116113745	1.0753479	0.026

6.3. DFS

Problema	Nós gerados	Nós expandidos	g(x) Profundidade	Profundidade máxima	Penetrância	ABF	Tempo(s)

((0 0 0 0 0 2) (0 0 0 0 4 0))	11	5	6	28	0.54545457	1.1762695	0.001
((2 2 2 2 2 2) (2 2 2 2 2 2))	95	17	18	28	0.18947369	1.1538696	0.003
((0 3 0 3 0 3) (3 0 3 0 3 0))	39	9	10	28	0.25641027	1.2473145	0.002
((1 2 3 4 5 6) (6 5 4 3 2 1))	Na	Na	Na	Na	Na	Na	Na
((2 4 6 8 10 12) (12 10 8 6 4 2))	Na	Na	Na	Na	Na	Na	Na
((48 0 0 0 0 0) (0 0 0 0 0 48))	Na	Na	Na	Na	Na	Na	Na
((8 8 8 8 8 8) (8 8 8 8 8 8))	Na	Na	Na	Na	Na	Na	Na