Ficha de laboratório Nº 8 e 9: Procura em espaço de estados com o algoritmo A* e resultados da procura

Inteligência Artificial - Escola Superior de Tecnologia de Setúbal

Prof. Joaquim Filipe
Eng. Filipe Mariano

Objetivos da ficha

Com este laboratório pretende-se que os alunos pratiquem a implementação de métodos de procura em espaço de estados, sendo os principais objectivos:

- Reforçar a aprendizagem da sintaxe da linguagem Lisp
- Implementar o algoritmo A* com uma heurística admissível
- Implementar a leitura e escrita de dados em ficheiros

1. O Problema das Vasilhas de Água

Alterar o programa desenvolvido para resolver o problema das vasilhas de água com os algoritmos BFS e DFS (ficha de laboratório nº 6), de modo a resolver o problema usando o algoritmo A* e calcular algumas métricas da procura.

1.1. Implementação do algoritmo A*

1. **Tipo abstrato de dados**: Alterar o tipo abstrato de dados definido anteriormente para o nó, de modo a que a terceira posição do mesmo represente o valor da heurística de um dado nó. A 4ª posição do tipo abstrato de dados passa a ser a representação do nó pai.

2. **Construtor**: Alterar a função no-teste por forma a que retorne uma lista com quatro elementos, em que o terceiro é o valor da heurística.

3. **Seletores**:

    - Redefinir a função `no-pai`, definida no laboratório 6, de modo a aceder agora ao quarto elemento do tipo abstrato de dados.

    - Definir uma função `no-heuristica`, que recebe um nó e retorna o valor da heurística do mesmo (terceiro elemento do tipo abstrato de dados). Definir uma função no-custo, que recebe um nó e retorna o custo do mesmo, com base na soma do valor da profundidade e da heurística do nó.

    - Alterar a função `cria-no` para que possa receber também o valor da heurística (agora o terceiro elemento do tipo abstrato de dados).

4. **Sucessores**: Alterar a função `sucessores` para que possa receber o nome da função a utilizar para calcular a heurística dos nós filhos do nó recebido como parâmetro.

5. **Ordenação dos nós**: Definir a função `ordenar-nos`, que implementa a ordenação crescente de uma lista de nós, de acordo com o valor do custo de cada nó.

6. **Heurística**: Definir uma função `heuristica`, que recebe um estado `n` e que retorna um valor inteiro, tal que:
   (`Va` representa o valor da vasilha `A` do nó e `Vb` representa o valor da vasilha `B`)

$$heuristica(n) = \begin{cases} 0 & \text{se } Va = 1 \text{ ou } Vb = 1 \\ 1 & \text{se } Va = Vb \text{ e } Va \neq 1 \\ 2 & \text{caso contrário} \end{cases}$$

7. **Ordenação para A***: Escrever uma função `colocar-sucessores-em-abertos`, que recebe duas listas. A primeira lista representa um conjunto de nós da lista de abertos, enquanto que a segunda lista representa o conjunto de nós sucessores de um determinado nó. A função retorna uma lista, resultante da junção de ambas e da sua ordenação por ordem crescente, com base no valor do custo de cada nó.

8. **Algoritmo A***: Defina a função `a*` que irá efetuar a procura recorrendo ao algoritmo A* e recebe como argumento o nó inicial, a função objetivo a ser testada, a função sucessores, a lista de operadores, a função heuristica e deverá retornar o nó solução encontrado.

   Ter em atenção, à função que verifica a existência do nó uma vez que apenas será possível expandir o nó caso não se encontre na lista de fechados um nó com o mesmo estado mas com um custo menor.

1.2. Apresentação da solução de procura

9. **Mostrar o resultado**: Criar uma função `mostrar-solucao`, que apresenta a solução encontrada pela procura, navegando no ramo que contém o nó final até ao nó raiz. A função poderá escrever, por exemplo, para cada nó que pertence ao caminho da solução, uma linha com o estado do problema, a profundidade, a heurística e o custo. A função mostra também o número de nós gerados e o número de nós expandidos.

1.3. Escrita em ficheiro

10. Definir uma função `escrever-no` que permite escrever num ficheiro `log.dat` o resultado de uma procura no espaço de estados. O resultado é composto pelo estado inicial, a solução encontrada, o número de nós gerados e o número de nós expandidos. A função deverá escrever no fim do ficheiro existente.

1.4. Interface com o utilizador

11. **Leitura do algoritmo de procura**: Alterar as funções `iniciar` e `ler-algoritmo` de modo a ler o nome da função heurística a utilizar na procura da solução, se o utilizador escolher a opção A*.

12. **Escrita dos resultados da procura**: Alterar a função `iniciar` de modo a escrever o resultado do procura da solução do problema no ficheiro `log.dat`.

1.4. Análise e Escrita de Resultados

13. **Tempo de Execução**: Alterar as funções dos algoritmos de modo a poderem retornar o tempo de execução de uma procura até encontrar a solução.

14. **Penetrância:** Definir uma função `penetrancia` que permite calcular a penetrância da árvore da procura. A função deverá receber o comprimento do caminho até ao objetivo (L) e o número total de nós gerados (T). Esta função será responsável por realizar o quociente entre o primeiro (L) e segundo (T) parâmetros que a função recebe.

15. **Fator de Ramificação Média:** Definir uma função que permite calcular o fator de ramificação média, através de uma função `bisseccao` que resolva a equação polinomial b^n+b^n-1+...+1=0 e que permite calcular o fator de ramificação médio de um grafo resultante de uma procura em espaço de estados.

16. Alterar a função `mostrar-solucao` para que passe a escrever, também, a penetrância, o fator de ramificação e o tempo de execução.

Leitura de um Estado Inicial

17. Criar um ficheiro `problema.dat` em que se coloque a seguinte lista `(0 0)`. Deve criar uma função que leia o ficheiro e após perguntar qual o algoritmo de procura, inicie a procura com o estado inicial lido a partir do ficheiro.