Relatório 1º Trabalho Prático - Inteligência Artifial

Bernardo Vitorino l48463, Daniel Barreiros l48452, Tomás Antunes l48511

24 de março de 2023

1 Labirinto - Agente A → Saida S

1.1 O problema

1.1.1 Espaço de resultados

De forma a retratar o problema em questão, representamos os estados na forma "`(X, Y)`", em que X é o número da linha e Y é o número da coluna de cada estada respetivamente. De forma a criar o estado inicial e final utilizámos este tipo de representação ficando na seguinte forma: `estado_inicial((6, 1))` e `estado_final((0, 4))`.

1.1.2 Restrições

De forma a criar os bloqueios utilizámos a mesma representação que nos estados inicial e final tendo sido criado um predicado que guarda as coordenadas dos mesmos.

- `bloqueado((0, 2))`
- `bloqueado((1, 0))`
- `bloqueado((1, 2))`
- `bloqueado((1, 6))`
- `bloqueado((3, 3))`
- `bloqueado((3, 4))`
- `bloqueado((3, 5))`

1.1.3 Operadores de transição de estado

Para representar as deslocações do agente (cima, baixo, esquerda, direita) utilizamos o predicado apresentado nas aulas praticas `op/4`. O predicado foi utilizado na forma `op(estado_atual, operador, estado_seguinte, custo)`. Este predicado valida o movimento do agente, verifica se a posição final do movimento não é um bloqueio, se excede os limites do problema ou se já foi percorrido.

```
nao_percorridos(A) :-
    \+ percorridos(A),
    assertz(percorridos(A)).

op((X, Y), sobe, (X, B), 1) :-
    tamanho(T), Y < T, B is Y + 1, \+ bloqueado((X,B)), nao_percorridos((X,B)).

op((X, Y), esquerda, (A, Y), 1) :-
    X > 0, A is X - 1, \+ bloqueado((A,Y)), nao_percorridos((A,Y)).

op((X, Y), desce, (X, B), 1) :-
    Y > 0, B is Y - 1, \+ bloqueado((X,B)), nao_percorridos((X,B)).

op((X, Y), direita, (A, Y), 1) :-
    tamanho(T), X < T, A is X + 1, \+ bloqueado((A,Y)), nao_percorridos((A,Y)).
```

1.2 Algoritmos de pesquisa não informada

De forma a obter o algoritmo de pesquisa não informada com melhor performance entre a pesquisa em profundidade e a pesquisa em largura fizemos o teste com os estados inicial e final apresentados no enunciado. O algoritmo que obteve o melhor resultado foi o algoritmo de pesquisa em profundidade apresentando os seguintes valores:

1.2.1 Análise dos algoritmos

Algoritmo	Estados Visitados	Estados em Memória	Profundidade	Custo
Profundidade	15	11	13	13

Tabela 1: Resultados obtidos utilizando o algoritmo de Pesquisa em Profundidade.

1.2.2 Código em Prolog

```
pesquisa_profundidade([no(E,Pai,Op,C,P)|_],no(E,Pai,Op,C,P)) :-
    estado_final(E), inc.

pesquisa_profundidade([E|R],Sol) :-
    inc, expande(E,Lseg),
    insere_inicio(Lseg,R,LFinal),
    length(LFinal, L), actmax(L),
    pesquisa_profundidade(LFinal,Sol).
```

1.3 Heurísitcas

1.3.1 Distância de Manhattan

Como primeira heurística foi utilizada a distância de Manhattan: $d((x1, y1), (x2, y2)) = |x1 - x2| + |y1 - y2|$.

```
distancia_manhattan((X, Y), (W, Z), D) :-
    X1 is X-W,
    abs(X1, AX),
    Y1 is Y-Z,
    abs(Y1, AY),
    D is AX+AY.
```

1.3.2 Distância euclidiana

Já a segunda heurística escolhida foi a distância euclidiana: $d((x1, y1), (x2, y2)) = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$.

```
euclidean_distance((X1, Y1), (X2, Y2), R) :-
    R is sqrt((X2-X1)^2 + (Y2-Y1)^2).
```

1.4 Algoritmos de pesquisa informada

A análise dos algoritmos de pesquisa informada foi realizada utilizando os estados inicial e final do enunciado.

1.4.1 Análise dos algoritmos

Após análise dos resultados obtidos pelos algoritmos A* e Greedy com as heurísticas mostradas anteriormente determinamos que o melhor algoritmo na resolução deste problema foi o Greedy, apresentado os seguintes resultados:

Algoritmo de Pesquisa	Estados Visitados	Estados em Memória	Profundidade	Custo
Greedy com dist. de Manhattan	39	53	13	13
Greedy com dist. euclidiana	9	13	9	9

Tabela 2: Resultados do Algoritmo Greedy.

1.4.2 Código em Prolog

```
pesquisa_g([no(E,Pai,Op,C,HC,P)|_],no(E,Pai,Op,C,HC,P)) :-
    estado_final(E).

pesquisa_g([E|R],Sol) :-
    inc, expande_g(E,Lseg),
    insere_ordenado(Lseg,R,Resto), length(Resto,N), actmax(N),
    pesquisa_g(Resto,Sol).
```

2 Labirinto - Agente A leva Caixa C → Saída S

2.1 O problema

2.1.1 Espaço de resultados

Para a representação do problema, os estados seguem todos a estrutura "`(X, Y, A, B)`", em que X é o número da coluna e Y o número da linha em que se encontra o agente e A é o número da coluna e B o número da linha em que se encontra a caixa. Através desta representação, foi possível criar os estados inicial e final: `estado_inicial((1, 6, 1, 5))` e `estado_final((_, _, 4, 0))`.

2.1.2 Restrições

As restrições seguem a mesma representação que o problema anterior e que os estados inicial e final tendo sido criada uma "lista"de cláusulas:

- `bloqueado((0, 1))`
- `bloqueado((2, 0))`
- `bloqueado((2, 1))`
- `bloqueado((3, 3))`
- `bloqueado((3, 4))`
- `bloqueado((3, 5))`
- `bloqueado((6, 1))`

2.1.3 Operadores de transição de estado

Para representar as deslocações do agente (cima, baixo, esquerda, direita) utilizamos o predicado apresentado nas aulas praticas `op/4`. O predicado foi utilizado na forma `op(estado_atual, operador, estado_seguinte, custo)`. Este predicado valida o movimento do agente, verifica se a posição final do movimento não é um bloqueio, se excede os limites do problema ou se já foi percorrido.

```
op((X, Y, A, B), cima, (X, Y1, A, B1), 1) :-
    Y1 is Y - 1,
    (iguais((X, Y1), (A, B)) -> (
            B1 is B - 1,
            lim(A, Y1),
            \+ bloqueado((A, B1))
        );
        (
            B1 is B,
            lim(X, Y1),
            \+ bloqueado((X, Y1))
        )
    ).

op((X, Y, A, B), direita, (X1, Y, A1, B), 1) :-
    X1 is X+1,
    (iguais((X1, Y), (A, B)) -> (
            A1 is A+1,
            lim(A1, B), lim(X1, Y),
            \+ bloqueado((A1, B))
        );
        (
            A1 is A,
            lim(X1, Y),
            \+ bloqueado((X1, Y))
        )
    ).
```

```
op((X, Y, A, B), baixo, (X, Y1, A, B1), 1) :-
    Y1 is Y+1,
    (iguais((X, Y1), (A, B)) -> (
            B1 is B+1,
            lim(A, B1), lim(X, Y1),
            \+ bloqueado((A, B1))
        );
        (
            B1 is B,
            lim(X, Y1),
            \+ bloqueado((X, Y1))
        )
    ).

op((X, Y, A, B), esquerda, (X1, Y, A1, B), 1) :-
    X1 is X-1,
    (iguais((X1, Y), (A, B)) -> (
            A1 is A-1,
            lim(A1, B), lim(X1, Y),
            \+ bloqueado((A1, B))
        );
        (
            A1 is A,
            lim(X1, Y),
            \+ bloqueado((X1, Y))
        )
    ).
```

2.2 Algoritmos de pesquisa não informada

Devido ao elevado número de nós que precisavam ser armazenados simultaneamente na memória tanto para o algoritmo de pesquisa em largura como para o algoritmo de pesquisa em profundidade decidimos utilizar um estado inicial diferente do estado inicial apresentado no enunciado. Optamos por realizar esta troca pois foi nos apresentado o erro `global stack overflow`. Assim sendo, utilizámos como estado inicial `(4, 6, 4, 5)`.

2.2.1 Análise dos algoritmos

Após a analise da performancedos dos algoritmos, nomeadamente os algoritmos de pesquisa em profundidade e pesquisa em largura, concluímos que o que obteve melhores resultados foi o algoritmo de pesquisa em profundidade.

Algoritmo	Estados Visitados	Estados em Memória	Profundidade	Custo
Profundidade	6	12	5	5

Tabela 3: Resultados obtidos através do algoritmo de Pesquisa em Profundidade.

2.2.2 Código em Prolog

```
pesquisa_profundidade([no(E,Pai,Op,C,P)|_],no(E,Pai,Op,C,P)) :-
    estado_final(E), inc.
pesquisa_profundidade([E|R],Sol) :-
    inc, expande(E,Lseg),
    insere_inicio(Lseg,R,LFinal),
    length(LFinal, L), actmax(L),
    pesquisa_profundidade(LFinal,Sol).
```

2.3 Heurísitcas

2.3.1 Distância de Manhattan entre a caixa C e a saída S - Heurística 1

Como primeira heurística, foi utilizada a distância de Manhattan entre as posições da caixa C e da saída S: $d((x1, y1), (x2, y2)) = |x1 - x2| + |y1 - y2|$.

```
distancia_manhattan((_, _, X, Y), (_, _, W, Z), D) :-
    X1 is abs(X - W),
    Y1 is abs(Y - Z),
    D is X1+Y1.
```

2.3.2 Distância Euclideana entre a caixa C e a saída S - Heurística 2

Já a segunda heurística escolhida foi a distância euclidiana entre a caixa C e a Saída S: $d((x1, y1), (x2, y2)) = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$.

```
euclidean_distance((_, _, X, Y), (_, _, A, B), R) :-
    R is sqrt((X-A)^2 + (Y-B)^2).
```

2.4 Algoritmos de pesquisa informada

A análise dos algoritmos de pesquisa informada foi realizada utilizando os estados inicial e final do enunciado. Após análise dos resultados obtidos através dos algoritmos A* e Greedy concluímos que o mais eficiente foi o algoritmo Greedy.

2.4.1 Análise dos algoritmos

Algoritmo de Pesquisa	Estados Visitados	Estados em Memória	Profundidade	Custo
Greedy com heurística 1	269	38	32	32
Greedy com heurística 2	269	38	32	32

Tabela 4: Análise dos algoritmos de Pesquisa Informada.

2.4.2 Código em Prolog

```
pesquisa_g(_, [no(E,Pai,Op,C,HC,P)|_], no(E,Pai,Op,C,HC,P)) :-
    estado_final(E), inc.
pesquisa_g(HEUR, [E|R], Sol) :-
    inc, expande_g(HEUR, E,Lseg), assertz(percorridos(E)),
    insere_ordenado(Lseg,R,Resto), length(Resto,N), actmax(N),
    pesquisa_g(HEUR, Resto, Sol).
```

3 Executar pesquisas

Para executar o programa:

- Carregar o problema desejado (labirinto ou labirinto2);
- Chamar o predicado responśavel pela pesquisa, utilizando pesquisa_p. (profundidade) pesquisa_g. (Greedy);