Resolução de problemas como problemas de pesquisa no espaço de estados

João Matos (32409), João Rouxinol (44451) e André Rato (45517)

24 de março de 2022

1 Agente A até à saída S

1.1 O problema

1.1.1 Espaço de resultados

Para a representação do problema, os estados seguem todos a estrutura `"e(C, L)"`, em que C é o número da coluna e L o número da linha do estado. Através desta representação, foi possível criar os estados inicial e final: `estado_inicial(e(2, 7))` e `estado_final(e(5, 1))`.

1.1.2 Restrições

As retrições seguem a mesma representação que os estados inicial e final, tido sido então criada uma "lista"de cláusulas:

- `bloqueada(e(1, 2))`
- `bloqueada(e(3, 1))`
- `bloqueada(e(3, 2))`
- `bloqueada(e(4, 4))`
- `bloqueada(e(4, 5))`
- `bloqueada(e(4, 6))`
- `bloqueada(e(7, 2))`

1.1.3 Operadores de transição de estado

De modo a representar as deslocações do agente para cima, baixo, esquerda e direita, o predicado `op/4` foi definido, utilizando a estrutura `op(estado_atual, operador, estado_seguinte, custo)`. Este predicado verifica se o movimento é válido no tamanho do problema, neste caso 7, e se a posição para qual o agente se movimenta está ou não bloqueada.

```
op(e(X, Y), cima, e(X, Y1), 1) :-
    Y > 1,
    Y1 is Y - 1,
    \+ bloqueada(e(X, Y1)).

op(e(X, Y), direita, e(X1, Y), 1) :-
    X < 7,
    X1 is X + 1,
    \+ bloqueada(e(X1, Y)).

op(e(X, Y), baixo, e(X, Y1), 1) :-
    Y < 7,
    Y1 is Y + 1,
    \+ bloqueada(e(X, Y1)).

op(e(X, Y), esquerda, e(X1, Y), 1) :-
    X > 1,
    X1 is X - 1,
    \+ bloqueada(e(X1, Y)).
```

Figura 1: Operadores de transição de estado.

1.2 Algoritmos de pesquisa não informada

A análise dos algoritmos de pesquisa não informada foi realizada com um estado inicial diferente do estado inicial do enunciado. Isto aconteceu pois devido ao elevado número de nós que necessitam de ser guardados simultanemante em memória para os algoritos de pesquisa em largura e em profundidade, é retornado o erro `global stack overflow`, informando que a memória disponibilizada para a execução do programa não é suficiente para armazenar os nós necessários. Sendo assim, o estado inicial utilizado foi `e(3, 5)`, não sendo alterado o estado final.

1.2.1 Análise dos algoritmos

Algoritmo	Estados Visitados	Estados em Memória	Profundidade	Custo
Largura	1045	1187	6	6
Profundidade	13	12	6	6
Profundidade Iterativa	56812	14	6	6

Tabela 1: Análise dos algoritmos de pesquisa não informada.

1.2.2 Algoritmo mais eficiente

Analisando os valores obtidos para os três algoritmos de pesquisa não informada, é possível constatar que:

- a **pesquisa em profundidade** possui menor número de estados visitados;
- a **pesquisa em profundidade** possui menor número de estados em memória;
- todos os algoritmos encontram uma solução para o problema à mesma profundidade e com o mesmo custo.

Após a análise acima, pode verificar-se que, embora o algoritmo de pesquisa em largura ser um algoritmo de pesquisa completo, este necessita de utilizar muita memória para guardar todos os nós. A pesquisa em profundidade iterativa é também um algoritmo completo, mas necessita de incrementar a profundidade a cada passo e refazer todas as árvores dos passos anteriores. Assim, mesmo não obtendo uma solução ótima, escolhe-se o algoritmo de pesquisa em profundidade.

1.2.3 Código em Prolog

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

1.3 Heurísitcas

1.3.1 Distância de Manhattan

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

```
manhattan(e(A,B),C):-
    estado_final(e(X,Y)),
    X1 is abs(A - X),
    Y1 is abs(B - Y),
    C is X1 + Y1.
```

Figura 2: Implementação da distância de Manhattan em Prolog.

1.3.2 Distância euclidiana

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

```
euclidiana(e(Ix,Iy),SOMA):-
    estado_final(e(Fx,Fy)),
    Dx is abs(Ix - Fx),
    Dy is abs(Iy - Fy),
    SOMA is round(sqrt(Dy ** 2 + Dx ** 2)).
```

Figura 3: Implementação da distância euclidiana em Prolog.

1.4 Algoritmos de pesquisa informada

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

1.4.1 Análise dos algoritmos

Algoritmo de Pesquisa	Estados Visitados	Estados em Memória	Profundidade	Custo
A* com dist. de Manhattan	60	43	9	9
A* com dist. euclidiana	66	39	9	9
Ansiosa com dist. de Manhattan	11	14	9	9
Ansiosa com dist. euclidiana	11	15	9	9

Tabela 2: Análise dos algoritmos de pesquisa informada.

1.4.2 Algoritmo mais eficiente

Analisando os valores obtidos para os três algoritmos de pesquisa informada, é possível constatar que:

- a **pesquisa ansiosa**, com qualqur uma das duas heurísticas, possui menor número de estados visitados;
- a **pesquisa ansiosa**, que utiliza como heurística a distância de Manhattan, possui menor número de estados em memória;
- todos os algoritmos encontram uma soluçao para o problema à mesma profundidade e com o mesmo custo.

Após a análise acima, pode verificar-se que o algoritmo com melhor desempenho foi o algoritmo de pesquisa ansiosa, sendo que o algoritmo que utiliza a distância de Manhattan necessitava de menos estados em memória para chegar à solução. Verificamos também que a pesquisa ansiosa expande o nó que aparenta estar mais próximo do estado final, o que neste caso, acaba por melhorar o desempenho do algoritmo.

1.4.3 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,
    asserta(fechado(E)),
    expande_g(E,Lseg),
    insere_ord(Lseg,R,Resto),
    length(Resto,N),
    actmax(N),
    pesquisa_g(Resto,Sol).
```

2 Agente A empurra a caixa C até à saída S

2.1 O problema

2.1.1 Espaço de resultados

Para a representação do problema, os estados seguem todos a estrutura "`p(C1, L1, C2, L2)`", em que C1 é o número da coluna e L1 o número da linha em que se encontra o agente e C2 é o número da coluna e L2 o número da linha em que se encontra a caixa. Através desta representação, foi possível criar os estados inicial e final: `estado_inicial(p(2, 7, 2, 6))` e `estado_final(p(_, _, 5, 1))`.

2.1.2 Restrições

As retrições seguem a mesma representação que o problema anterior e que os estados inicial e final, tido sido então criada uma "lista"de cláusulas:

- `bloqueada(p(1, 2))`
- `bloqueada(p(3, 1))`
- `bloqueada(p(3, 2))`
- `bloqueada(p(4, 4))`
- `bloqueada(p(4, 5))`
- `bloqueada(p(4, 6))`
- `bloqueada(p(7, 2))`

2.1.3 Operadores de transição de estado

De modo a representar as deslocações do agente para cima, baixo, esquerda e direita, e o movimento da caixa, o predicado `op/4` foi definido, utilizando a estrutura `op(estado_atual, operador, estado_seguinte, custo)`.

```
op(p(X, Y, P, Q), cima, p(X, Y1, P, Q1), 1) :-
    Y1 is Y - 1,
    (iguais(p(X, Y1), p(P, Q)) ->
        (
            Q1 is Q - 1,
            lim(P, Y1),
            \+ bloqueada(p(P, Q1))
        );
        (
            Q1 is Q,
            lim(X, Y1),
            \+ bloqueada(p(X, Y1))
        )
    ).
```

```
op(p(X,Y,P,Q),direita,p(X1,Y,P1,Q),1) :-
    X1 is X+1,
    (iguais(p(X1,Y),p(P,Q)) ->
        (
            P1 is P+1,
            lim(P1,Q),
            lim(X1,Y),
            \+ bloqueada(p(P1,Q))
        );
        (
            P1 is P,
            lim(X1,Y),
            \+ bloqueada(p(X1,Y))
        )
    ).
```

```
op(p(X,Y,P,Q),baixo,p(X,Y1,P,Q1),1) :-
    Y1 is Y+1,
    (iguais(p(X,Y1),p(P,Q)) ->
        (
            Q1 is Q+1,
            lim(P,Q1),
            lim(X,Y1),
            \+ bloqueada(p(P,Q1))
        );
        (
            Q1 is Q,
            lim(X,Y1),
            \+ bloqueada(p(X,Y1))
        )
    ).
```

```
op(p(X,Y,P,Q),esquerda,p(X1,Y,P1,Q),1) :-
    X1 is X-1,
    (iguais(p(X1,Y),p(P,Q)) ->
        (
            P1 is P-1,
            lim(P1,Q),
            lim(X1,Y),
            \+ bloqueada(p(P1,Q))
        );
        (
            P1 is P,
            lim(X1,Y),
            \+ bloqueada(p(X1,Y))
        )
    ).
```

Figura 4: Operadores de transição de estado.

2.2 Algoritmos de pesquisa não informada

A análise dos algoritmos de pesquisa não informada foi realizada com um estado inicial diferente do estado inicial do enunciado. Isto aconteceu pois devido ao elevado número de nós que necessitam de ser guardados simultanemante em memória para os algoritos de pesquisa em largura e em profundidade, é retornado o erro `global stack overflow`, informando que a memória disponibilizada para a execução do programa não é suficiente para armazenar os nós necessários. Sendo assim, o estado inicial utilizado foi `p(5, 7, 5, 6)`, não sendo alterado o estado final.

2.2.1 Análise dos algoritmos

Algoritmo	Estados Visitados	Estados em Memória	Profundidade	Custo
Largura	213	216	5	5
Profundidade	11	12	5	5
Profundidade Iterativa	6281	12	5	5

Tabela 3: Análise dos algoritmos de pesquisa não informada.

2.2.2 Algoritmo mais eficiente

Analisando os valores obtidos para os três algoritmos de pesquisa não informada, é possível constatar várias coisas:

- a **pesquisa em profundidade** possui menor número de estados visitados;
- as **pesquisas em profudidade** e `em profundidade iterativa` possuem menor número de estados em memória;
- todos os algoritmos encontram uma soluçao para o problema à mesma profundidade e com o mesmo custo.

Após a análise acima, pode verificar-se que o algoritmo com melhor desempenho foi, novamente, o algoritmo de pesquisa em profundidade. No entanto, é possível afirmar-se que o algoritmo de pesquisa em profundidade iterativa tem um grande nível de abrangência no que toca à gestão da memória disponibilizada, sendo, dos três algoritmos, aquele que guarda simultaneante, menos estados em meória.

2.2.3 Código em Prolog

```
pesquisa_profundidade([no(E,Pai,Op,C,P)|_],no(E,Pai,Op,C,P)):-
    inc,
    estado_final(E).
pesquisa_profundidade([E|R],Sol):-
    inc,
    expande(E,Lseg),
    insere_fim(R,Lseg,Resto),
    length(Resto,N),
    actmax(N),
    pesquisa_profundidade(Resto,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: $d((x1, y1), (x2, y2)) = |x1 - x2| + |y1 - y2|$.

```
dist_estados(p(_,_,Ix,Iy),SOMA):-
    estado_final(p(_,_,Fx,Fy)),
    Dx is abs(Ix - Fx),
    Dy is abs(Iy - Fy),
    SOMA is Dx + Dy.
```

Figura 5: Implementação da distância de Manhattan entre a caixa C e a saída S em Prolog.

2.3.2 Diferença entre as linhas da caixa C e a saída C - heurística 2

Já a segunda heurística escolhida foi a diferença entre o valor da linha de C e o valor da linha de S: $d((x1, y1), (x2, y2)) = |y1 - y2|$.

```
euclidiana(e(Ix,Iy),SOMA):-
    estado_final(e(Fx,Fy)),
    Dx is abs(Ix - Fx),
    Dy is abs(Iy - Fy),
    SOMA is round(sqrt(Dy ** 2 + Dx ** 2)).
```

Figura 6: Implementação da distância euclidiana em Prolog.

2.4 Algoritmos de pesquisa informada

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

2.4.1 Análise dos algoritmos

Algoritmo de Pesquisa	Estados Visitados	Estados em Memória	Profundidade	Custo
A* com heurística 1	1203	459	16	16
A* com heurística 2	2305	513	16	16
Ansiosa com heurística 1	467	54	16	16
Ansiosa com heurística 2	2097	198	16	16

Tabela 4: Análise dos algoritmos de pesquisa informada.

2.4.2 Algoritmo mais eficiente

Analisando os valores obtidos para os três algoritmos de pesquisa informada, é possível constatar que:

- a **pesquisa ansiosa com a heurística 1** possui menor número de estados visitados;
- a **pesquisa ansiosa com a heurística 1** possui menor número de estados em memória;
- todos os algoritmos encontram uma soluçao para o problema à mesma profundidade e com o mesmo custo.

Podemos então dizer que o melhor algoritmo de pesquisa informada para este problema é o algoritmo de pesquisa ansiosa em que a heurística utilizada é a distâcia de Manhattan entre a caixa C e a saída S, mostrando ter um melhor desempenho geral, não visitando tantos nós, nem utilizando tanto memória como os demais algoritmos.

2.4.3 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,
    asserta(fechado(E)),
    expande_g(E,Lseg),
    insere_ord(Lseg,R,Resto),
    length(Resto,N),
    actmax(N),
    pesquisa_g(Resto,Sol).
```

3 Executar pesquisas

Para executar o programa basta:

- carregar o problema desejado e o tipo de pesquisa que é pretendido ("pi", para pesquisas informadas, ou "pni", para pesquisas não informadas, utilizando `[<problema>, <tipo_pesquisa>].`;
- chamar o predicado responśavel pela pesquisa, utilizando `pesquisa(<problema>, <algoritmo>).`;
- esperar que o programa encontre a solução para o problema pretendido e observar os resultados obtidos.