Języki Przetwarzania Symbolicznego
Projekt : Modyfikacja algorytmu A*

Stawczyk Przemysław

1 Opis zadania

Wprowadzeniu do programu zmian implementujących program do testowego wykonania algorytmu wg następującej zasady:

Zadane jest ograniczenie liczby kroków – w postaci dodatkowego argumentu procedury *search_ A_ star*.

W każdym kroku algorytm pobiera z kolejki (uporządkowanej wg oceny heurystycznej stanu), *bez usuwania*, pierwsze N węzłów – N jest zadane w wywołaniu procedury *search_ A_ star* (w postaci dodatkowego argumentu). Lista N węzłów jest przedstawiana użytkownikowi (wyprowadzana używając *write*), wraz z informacją, który jest to krok wykonania algorytmu. Użytkownik podaje listę numerów, która określa *kolejność pobierania węzłów z listy węzłów do kolejnych testowych przebiegów dalszego wykonania algorytmu.* Nazwijmy tę listę numerów listą wyboru. Do dalszego przebiegu wykonania zostaje pobrany węzeł wskazany pierwszym numerem na liście wyboru.

W każdym kroku jest badany i aktualizowany licznik kroków. Jeśli w którymś kroku zostanie wyczerpany limit liczby kroków, to algorytm nawraca do poprzedniego kroku , aby wybrać na tym poziomie rekurencji kolejny węzeł z listy węzłów – *zgodnie z kolejnością podaną na liście wyboru.* Po wyczerpaniu możliwości wyboru węzła w tym kroku, algorytm pyta użytkownika, czy zwiększyć limit:

- w przypadku odpowiedzi *tak* limit zostaje zwiększony i wykonanie jest kontynuowane (a nie rozpoczynane od nowa) z nowym ograniczeniem
- w przypadku odpowiedzi *nie* algorytm nawraca do wcześniejszego kroku w celu wyboru innego węzła wg listy wyboru

(odpowiedź użytkownika jest czytana za pomocą *read*, użytkownik wprowadza sekwencję <TEKST> <KROPKA> <ENTER>) W ramach zadanego ograniczenia liczby kroków są więc badane wszystkie kombinacje możliwych wyborów węzłów na poszczególnych poziomach rekurencji–na każdym poziomie w kolejności ustalonej numerami na liście wyboru.

2 Rozwiązanie

2.1 Wynikowy kod

```
start_A_star(InitState, PathCost, N, StepLimit):-
    score(InitState, 0, 0, InitCost, InitScore),
    search_A_star([node(InitState, nil, nil, InitCost , InitScore )],
                  [], PathCost, N, 1, StepLimit).
```

```
search_A_star(Queue, ClosedSet, PathCost, N, StepCounter, StepLimit):-
    write("search_A_star - Krok nr. : "), write(StepCounter), nl,
    fetch_new(Node, Queue, ClosedSet, RestQueue, N),
    continue(Node, RestQueue, ClosedSet, PathCost, N, StepCounter, StepLimit).

search_A_star(Queue, ClosedSet, PathCost, N, StepCounter, StepLimit):-
    write('search_A_star - Osiągnięto limit krokow. Zwiekszyc limit? (t/n)'), nl,
    read('t'),
    NewLimit is StepLimit + 1,
    search_A_star(Queue, ClosedSet, PathCost, N, StepCounter, NewLimit).

continue(node(State, Action, Parent, Cost, _), _,
        ClosedSet, path_cost(Path, Cost), _, _, _):-
    write('continue - check goal : '), write(State), nl,
    goal(State),!,
    write('continue - reached goal'), nl,
    build_path(node(Parent, _, _, _, _), ClosedSet, [Action/State], Path).

continue(Node, RestQueue, ClosedSet, Path, N, StepCounter, StepLimit):-
    StepCounter < StepLimit,
    write('continue '), write(StepCounter),
    write(' expand : '), write(Node), nl,
    expand(Node, NewNodes),
    insert_new_nodes(NewNodes, RestQueue, NewQueue),
    NewStepCounter is StepCounter + 1,
    search_A_star(NewQueue, [Node| ClosedSet], Path,
                  N, NewStepCounter, StepLimit).

fetch_new(Node, Queue, ClosedSet, RestQueue, N):-
    get_user_decisions(Queue, N, ClosedSet, Decisions),
    get_index(Index, Decisions),
    get_element_at_index(Index, Node, Queue, ClosedSet, RestQueue).

get_user_decisions(Queue, N, ClosedSet, Decisions):-
    output_nodes(Queue, N, ClosedSet, Diff, 1),
    write('Podaj indeksy wezlow: '), nl,
    VariantsNo is N - Diff,
    input_decisions(VariantsNo, Decisions).

output_nodes(_, 0, _, 0, _):- ! .

output_nodes([], N, _, N, _).

output_nodes([node(State, _, _, _, _)|R], N, ClosedSet, N2, F):-
    member(node(State, _, _, _, _), ClosedSet), !,
    output_nodes(R, N, ClosedSet, N2, F).

output_nodes([X|R], N, ClosedSet, N2, F):-
    write(F), write(' - '), write(X), nl,
    NewN is N - 1,
```

```
    NewF is F + 1,
    output_nodes(R, NewN, ClosedSet, N2, NewF).

input_decisions(0, []):- ! .

input_decisions(N, [D|RestDecisions]):-
    read(D),
    NewN is N - 1,
    input_decisions(NewN, RestDecisions).

get_index(X, [X|_]).

get_index(X, [_|R]):-
    get_index(X, R).

get_element_at_index(1, node(State, Action, Parent, Cost, Score),
                     [node(State, Action, Parent, Cost, Score)|R],
                     ClosedSet, R):-
    \+ member(node(State, _, _, _, _), ClosedSet), ! .

get_element_at_index(Index, Node,
                     [node(State, Action, Parent, Cost, Score)|R],
                     ClosedSet,
                     [node(State, Action, Parent, Cost, Score)|RestQueue]):-
    \+ member(node(State, _, _, _, _), ClosedSet), ! ,
    NewIndex is Index - 1,
    get_element_at_index(NewIndex, Node, R, ClosedSet, RestQueue).

get_element_at_index(Index, Node, [X|R], ClosedSet, [X|RestQueue]):-
    get_element_at_index(Index, Node, R, ClosedSet, RestQueue).

expand(node(State, _ ,_ , Cost, _ ), NewNodes):-
    findall(node(ChildState, Action, State, NewCost, ChildScore),
            (succ(State, Action, StepCost, ChildState),
                  score(ChildState, Cost, StepCost, NewCost, ChildScore)),
            NewNodes), ! .

score(State, ParentCost, StepCost, Cost, FScore):-
    Cost is ParentCost + StepCost,
    hScore(State, HScore),
    FScore is Cost + HScore.

insert_new_nodes( [ ], Queue, Queue) .

insert_new_nodes( [Node|RestNodes], Queue, NewQueue):-
    insert_p_queue(Node, Queue, Queue1),
    insert_new_nodes( RestNodes, Queue1, NewQueue) .

insert_p_queue(Node,  [ ], [Node] ):- ! .

insert_p_queue(node(State, Action, Parent, Cost, FScore),
```

```
        [node(State1, Action1, Parent1, Cost1, FScore1)|RestQueue],
        [node(State1, Action1, Parent1, Cost1, FScore1)|Rest1]):-
    FScore >= FScore1, ! ,
    insert_p_queue(node(State, Action, Parent, Cost, FScore), RestQueue, Rest1).

insert_p_queue(node(State, Action, Parent, Cost, FScore), Queue,
        [node(State, Action, Parent, Cost, FScore)|Queue]).

build_path(node(nil, _, _, _, _ ), _, Path, Path):- ! .

build_path(node(EndState, _ , _ , _, _ ), Nodes, PartialPath, Path):-
    del(Nodes, node(EndState, Action, Parent , _ , _  ) , Nodes1) ,
    build_path(node(Parent,_ ,_ , _ , _ ), Nodes1,
               [Action/EndState|PartialPath],Path).

del([X|R],X,R).

del([Y|R],X,[Y|R1]):-
    X\=Y,
    del(R,X,R1).
```

2.2 Przestrzeń stanów

```
% example grid
succ(a, a-b, 2, b).
succ(a, a-c, 3, c).
succ(b, b-g, 4, g).
succ(b, b-f, 3, f).
succ(c, c-d, 2, d).
succ(c, c-e, 3, e).
succ(g, g-m, 2, m).
succ(f, f-h, 4, h).
succ(d, d-m, 2, m).
succ(e, e-m, 5, m).

% set goal
goal(m).

% basic heuristic
hScore(a, 4).
hScore(b, 4).
hScore(c, 3).
hScore(d, 1).
hScore(e, 4).
hScore(f, 7).
hScore(g, 1).
hScore(h, 3).
hScore(m, 0).
```

3 Ślad wykonania

Ślad wykonania dla procedury: *start_A_star(a, R, 3, 4).*

Wyjście:

```
search_A_star - Krok nr. : 1
1 - node(a, nil, nil, 0, 4)
Podaj indeksy wezlow:
```

Dostępny jest do wyboru jedynie węzeł startowy - *a*.
Wejście:

```
1.
```

Wyjście:

```
continue - check goal : a
continue 1 expand : node(a, nil, nil, 0, 4)
search_A_star - Krok nr. : 2
1 - node(b, a-b, a, 2, 6)
2 - node(c, a-c, a, 3, 6)
Podaj indeksy wezlow:
```

Próba dopasowania do węzła terminalnego, a następnie rozwinięcie węzła.
Wejście:

```
2.
1.
```

Wyjście:

```
continue - check goal : c
continue 2 expand : node(c, a-c, a, 3, 6)
search_A_star - Krok nr. : 3
1 - node(b, a-b, a, 2, 6)
2 - node(d, c-d, c, 5, 6)
3 - node(e, c-e, c, 6, 10)
Podaj indeksy wezlow:
```

Sprawdzany jest pierwszy węzeł z listy wyboru - *node(c, a-c, a, 3, 6).*
Wejście:

```
2.
3.
1.
```

Wyjście:

```
continue - check goal : d
continue - check goal : e
continue - check goal : b
search_A_star - Osiągnięto limit krokow. Zwiekszyc limit? (t/n)
```

Sprawdzane są wszystkie możliwości na tym poziomie w zadanej kolejności, jako, że osiągnięty został limit kroków. Następnie następuje pytanie do użytkownika - udzielamy odpowiedzi odmownej.

Wejście:

```
n.
```

Wyjście:

```
continue - check goal : b
continue 2 expand : node(b, a-b, a, 2, 6)
search_A_star - Krok nr. : 3
1 - node(c, a-c, a, 3, 6)
2 - node(g, b-g, b, 6, 7)
3 - node(f, b-f, b, 5, 12)
Podaj indeksy wezlow:
```

Następuje wycofanie się do poziomu niżej i rozwinięcie kolejnego węzła - node(b, a-b, a, 2, 6).
Wejście:

```
2.
3.
1.
```

Wyjście:

```
continue - check goal : g
continue - check goal : f
continue - check goal : c
search_A_star - Osiągnięto limit krokow. Zwiekszyc limit? (t/n)
```

Sprawdzane są wszystkie możliwości na tym poziomie jako, że osiągnięty został limit kroków. Następnie następuje pytanie do użytkownika - udzielamy zgody.
Wejście:

```
t.
```

Wyjście:

```
search_A_star - Krok nr. : 3
1 - node(c, a-c, a, 3, 6)
2 - node(g, b-g, b, 6, 7)
3 - node(f, b-f, b, 5, 12)
Podaj indeksy wezlow:
```

Rozwiązywanie jest kontynuowane z nowym limitem.
Wejście:

```
3.
2.
1.
```

Wyjście:

```
continue - check goal : f
continue 3 expand : node(f, b-f, b, 5, 12)
search_A_star - Krok nr. : 4
1 - node(c, a-c, a, 3, 6)
2 - node(g, b-g, b, 6, 7)
3 - node(h, f-h, f, 9, 12)
Podaj indeksy wezlow:
```

Przechodzimy do poziomu 4. Rozwijany jest wybrany węzeł.
Wejście:

```
3.
2.
1.
```

Wyjście:

```
continue - check goal : h
continue - check goal : g
continue - check goal : c
search_A_star - Osiągnięto limit krokow. Zwiekszyc limit? (t/n)
```

Ponownie osiągamy limit głębokości wywołania, sprawdzając wcześniej możliwości z tego poziomu. Odmawiamy zwiększenia limitu.
Wejście:

```
n.
```

Wyjście:

```
continue - check goal : g
continue 3 expand : node(g, b-g, b, 6, 7)
search_A_star - Krok nr. : 4
1 - node(c, a-c, a, 3, 6)
2 - node(m, g-m, g, 8, 8)
3 - node(f, b-f, b, 5, 12)
Podaj indeksy wezlow:
```

Wejście:

```
2.
1.
3.
```

Wyjście:

```
continue - check goal : m
continue - reached goal
R = path_cost([nil/a, (a-b)/b, (b-g)/g, (g-m)/m], 8)
```

Sprawdzony węzeł prowadzi do stanu docelowego - rozwinięcie ścieżki od węzła startowego do terminalnego.