Государственное образовательное учреждение высшего
профессионального образования
“Московский государственный технический университет имени
Н.Э.Баумана”

ОТЧЕТ

По индивидуальному заданию
На тему «Реализация алгоритма A* на Prolog»
По курсу «Функциональное и логическое программирование»

Студент: Зыкин Д.А.
Группа: ИУ7 63
Преподаватель: Толпинская Н. Б.

2019

Введение

Как и алгоритм поиска в ширину, алгоритм A* является полным в том смысле, что он всегда находит решение, если таковое существует.

Если эвристическая функция h допустима, т. е. никогда не переоценивает действительную минимальную стоимость достижения цели, то алгоритм A* сам является допустимым (или оптимальным) также при условии, что мы не отсекаем пройденные вершины. Если же мы это делаем, то для оптимальности алгоритма требуется, чтобы h(x) была еще и монотонной или преемственной эвристикой. Свойство монотонности означает, что если существуют пути A–B–C и A–C (необязательно через B), то оценка стоимости пути от A до C должна быть меньше суммы оценок путей A–B и B–C либо равна этой сумме. (Монотонность также известна как неравенство треугольника: одна сторона треугольника не может быть длиннее, чем сумма двух других сторон.) Математически для всех путей x, y (где y – потомок x) выполняется условие

$$g(x) + h(x) <= g(y) + h(y).$$

Алгоритм A* также оптимально эффективен для заданной эвристики h. Это значит, что любой другой алгоритм исследует не меньше узлов, чем A* (за исключением случаев, когда существует несколько частных решений с одинаковой эвристикой, точно соответствующей стоимости оптимального пути).

Допустимость и оптимальность алгоритма

Алгоритм A* и допустим, и обходит при этом минимальное количество вершин, благодаря тому что работает с «оптимистичной» оценкой пути через вершину. Оптимистичной в том смысле, что если он пойдет через эту вершину, то у него «есть шанс», что реальная стоимость результата будет равна этой оценке, но никак не меньше. Поскольку A* является информированным алгоритмом, такое равенство может быть вполне возможным.

Когда алгоритм A* завершает поиск, то, согласно определению, он находит путь, истинная стоимость которого меньше, чем оценка стоимости любого пути через любой открытый узел. Но поскольку эти оценки являются оптими стичными, соответствующие узлы можно без сомнений отбросить. Иначе говоря, A* никогда не упустит возможности минимизировать длину пути и потому является допустимым.

Предположим теперь, что некий алгоритм Б вернул в качестве результата путь, длина которого больше оценки стоимости пути через некоторую вершину. На основе эвристической информации для алгоритма Б нельзя исключить возможность, что этот путь имел и меньшую реальную длину, чем результат. Соответственно, пока алгоритм Б просмотрел меньше вершин, чем A*, он не будет допустимым. Итак, A* проходит наименьшее количество вершин графа среди допустимых алгоритмов, использующих такую же точную (или менее точную) эвристику.

Оценка сложности

Временная сложность алгоритма A* зависит от эвристики. В худшем случае число вершин, исследуемых алгоритмом, растет экспоненциально по сравнению с длиной оптимального пути, но сложность становится полиномиальной, когда эвристика удовлетворяет следующему условию:

$$|h(x) - h^*(x)| <= O(\log h^*(x)),$$

где h* – оптимальная эвристика, т. е. точная оценка расстояния из вершины x к цели. Другими словами, ошибка h(x) не должна расти быстрее, чем логарифм от оптимальной эвристики.

АЛГОРИТМ

1. Создается два списка вершин – ожидающие рассмотрения и уже рассмотренные. В ожидающие добавляется точка старта, список рассмотренных пока пуст.
2. Для каждой точки рассчитывается F = G + H, где G – расстояние от старта до точки, H – примерное расстояние от точки до цели. Также каждая точка хранит ссылку на точку, из которой в нее пришли.
3. Из списка точек на рассмотрение выбирается точка с наименьшим F. Обозначим ее X.
4. Если X – цель, то мы нашли маршрут.
5. Переносим X из списка ожидающих рассмотрения в список уже рассмотренных.
6. Для каждой из точек, соседних для X (обозначим эту точку Y), делаем следующее:
   - если Y уже находится в рассмотренных – пропускаем ее;
   - если Y еще нет в списке на ожидание – добавляем ее туда, запомнив ссылку на X и рассчитав Y.G (это X.G + расстояние от X до Y) и Y.H;
   - если же Y в списке на рассмотрение – проверяем, если X.G + расстояние от X до Y < Y.G, значит мы пришли в точку Y более коротким путем, заменяем Y.G на X.G + расстояние от X до Y, а точку, из которой пришли в Y, на X;
   - если список точек на рассмотрение пуст, а до цели мы так и не дошли – значит маршрут не существует;

Реализация на SWI-Prolog

```
city(1,'City A').
city(2,'City B').
city(3,'City C').
city(4,'City D').
city(5,'City E').
city(6,'City F').
city(7,'City G').

road(1,2,436).
road(1,7,600).
road(1,3,78).
road(2,4,399).
road(2,5,85).
road(3,7,260).
road(3,6,227).
road(3,4,175).
road(5,7,241).
road(4,6,390).
road(4,5,481).

h(1,300).
h(2,250).
h(3,250).
h(4,100).
h(5,80).
h(6,50).
h(7,50).
h(8,200).

highway(1,2,'M12').
highway(1,3,'M13').
highway(2,4,'M24').
highway(4,5,'M$%').
highway(3,4,'M34').
highway(3,7,'M37').
highway(3,6,'M36').
highway(5,7,'M57').
highway(2,5,'M25').
highway(4,6,'M46').
```

```
/*
 * find_shortest_path
 * Этот предикат реализует алгоритм A * для графа, определенного ранее.
 * Находит кратчайший путь между пунктом отправления и пунктом назначения.
*/
find_shortest_path(Origin, Destination):-
        city(C1,Origin),
        city(C2,Destination),
        a_star([[0,C1]],C2,ReversePath),
        reverse(ReversePath, Path),
        write('The best/shortest Path is: '), print_path(Path,Highways),
       write('Highway to be traveled will be: '),print_highways(Highways),!.
/*
 * Это сообщение будет отображаться, если начало или пункт назначения не были
 * введены правильно
*/
find_shortest_path(_,_):- write('There was an error with origin or destination city, please type again').

/*
 * find_all
 * Этот предикат действует так же, как и предыдущий, но он продолжает искать
 * и показывает все пути
*/
find_all(Origin, Destination):-
        city(C1,Origin),
        city(C2,Destination),
        a_star([[0,C1]],C2,ReversePath),
        reverse(ReversePath, Path),
        write('A Path was found: '), print_path(Path,Highways),
        write('Highway to be traveled will be: '),print_highways(Highways),fail.
/*
 * Это сообщение будет отображаться, если начало или пункт назначения не были
 * введены правильно
*/
find_all(_,_):- write('That is all!').

/*
 * a_star
 * Мы применяем алгоритм поиска A *.
 * Начиная с начального узла, мы строим возможные пути
 * используя вспомогательные предикаты, определенные ниже.
*/
a_star(Paths, Dest, [C,Dest|Path]):-
```

```
        member([C,Dest|Path],Paths),
        decide_best(Paths, [C1|_]),
        C1 == C.
a_star(Paths, Destination, BestPath):-
        decide_best(Paths, Best),
        delete(Paths, Best, PreviousPaths),
        expand_border(Best, NewPaths),
        append(PreviousPaths, NewPaths, L),
        a_star(L, Destination, BestPath).

/*
 * decide_best
 * Из всех новых дорог, расширенных границей, мы выбираем ту,
 * у которой наилучшее f = h + g..
 * Мы сравниваем каждый первый элемент пути (то есть затраты)
 * и к этому добавляем соответствующую эвристику предпоследнего элемента
 * списка (то есть текущего города).
 * Это дает нам f, и мы выбираем путь лучшего f.
*/
decide_best([X],X):-!.
decide_best([[C1,Ci1|Y],[C2,Ci2|_]|Z], Best):-
        h(Ci1, H1),
        h(Ci2, H2),
        H1 +  C1 =< H2 +  C2,
        decide_best([[C1,Ci1|Y]|Z], Best).
decide_best([[C1,Ci1|_],[C2,Ci2|Y]|Z], Best):-
        h(Ci1, H1),
        h(Ci2, H2),
        H1  + C1 > H2 +  C2,
        decide_best([[C2,Ci2|Y]|Z], Best).

/*
 * expand_border
 * Мы ищем все узлы графа, которые являются границей текущего города,
 * и привязываемся к списку L.
 * Затем мы обновляем стоимость всех новых дорог, которые находятся в L.
*/
expand_border([Cost,City|Path],Paths):-
        findall([Cost,NewCity,City|Path],
                (road(City, NewCity,_),
                not(member(NewCity,Path))),
                L),
        change_costs(L, Paths).
```

```
/*
 * change_costs
 * Я обновляю стоимость каждого пути после добавления последнего узла.
 * Нужно помнить, что каждый путь представлен списком, где первый элемент - это
 * общая стоимость дороги, а следующие элементы очереди - это индексы городов
*/
change_costs([],[]):-!.
change_costs([[Total_Cost,Ci1,Ci2|Path]|Y],[[NewCost_Total,Ci1,Ci2|Path]|Z]):-
        road(Ci2, Ci1, Distance),
        NewCost_Total is Total_Cost + Distance,
        change_costs(Y,Z).

/* Печать пути */
print_path([Cost],[]):- nl, write('The total cost of the path is: '), write(Cost), write(' kilometers'),nl.
print_path([City,Cost],[]):- city(City, Name), write(Name), write(' '), nl, write('The total cost of the path is: '), write(Cost), write(' kilometers'),nl.
print_path([City,City2|Y],Highways):-
        city(City, Name),
        highway(City,City2,Highway),
        append([Highway],R,Highways),
        write(Name),write(', '),
        print_path([City2|Y],R).

/* Печать маршрутов */
print_highways([X]):- write(X), nl, nl.
print_highways([X|Y]):-
        write(X),write(' - '),
        print_highways(Y).
```

Результат

h177:Prolog zykinda$ swipl astar.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- find_shortest_path('City A','City E').
The best/shortest Path is: City A, City B, City E
The total cost of the path is: 521 kilometers
Highway to be traveled will be: M12 - M25

true.

h177:Prolog zykinda$ swipl astar.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- find_shortest_path('City A','City G').
The best/shortest Path is: City A, City C, City G
The total cost of the path is: 338 kilometers
Highway to be traveled will be: M13 - M37

true.

h177:Prolog zykinda$ swipl astar.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- find_shortest_path('City A','City A').
The best/shortest Path is: City A
The total cost of the path is: 0 kilometers
Highway to be traveled will be: There was an error with origin or destination city, please type again
true.