Студент: Красоткин Семён
Группа: М80-208Б-19
Вариант: 3

Лабораторная работа №3: Применение Пролога для решения задач поиска в пространстве состояний

Введение

Поиск в пространстве состояний — группа математических методов, предназначенных для решения задач искусственного интеллекта.

Методы поиска в пространстве состояний заключаются в последовательном просмотре конфигураций или состояний задачи с целью обнаружения целевого состояния, имеющего заданные характеристики или удовлетворяющего некоторому критерию.

Лучше всего метод поиска в пространстве состояний работает с задачами, в которых, очевидно, можно определить это пространство, а также начальные и конечные состояния, переходы от одного состояния к другому. Таким образом, это те задачи, которые можно свести к поиску решений в графе.

Что касается Prolog то, в этом языке очень удобно задавать состояния в виде предикатов. Сам Prolog использует один из методов поиска — поиск в глубину. Это делает использование данного метода простым и естественным, хоть и не оптимальным.

Задание

Железнодорожный сортировочный узел устроен так, как показано на рисунке (см. ЛР-практикум). На левой стороне собрано некоторое число вагонов двух типов (черные и белые), обоих типов по n штук., в произвольном порядке Тупик вмещает все 2n вагонов. Пользуясь тремя сортировочными операциями (слева в тупик, из тупика направо, слева направо, минуя тупик), собрать вагоны на правой стороне, так, чтобы типы чередовались. Для решения задачи достаточно 3n-1 сортировочных операций.

Принцип решения

В решении использованы 3 алгоритма поиска:

- В глубину (предикат *search_deep*);
- В ширину (предикат *search_bridth*);
- С итеративным погружением (предикат *search_id*).

Для каждого алгоритма-предиката есть общий предикат *prolong*() для для продления всех путей в графе, дабы не было зацикливания.

```
prolong(Given,Given1,[H|T],[Y,H|T]):-
   move(Given,H,Given1,Y),
   \+ member(Y,[H|T]).
```

Предикат поиска в глубину *search_deep*() работает, пока возможно продление пути и не достигнута конечная вершина. Путь записан в обратном порядке, поэтому он реверсируется, как и для следующих алгоритмов. Причём найденный путь не всегда кратчайший.

```
search_deep(A,B) :-
   write('DFS start for': A), nl,
   get_time(DFS_START),
   deep([[]],B,L,[],A),
   inv_print(L),
   get_time(DFS_END),
   write('DFS END'), nl, nl,
   T1 is DFS_END - DFS_START,
   write('DFS time: '), write(T1), nl, nl.
```

search_bridth() ищет решение в ширину, используя очередь из путей, которые можно продлить. Продленные пути добавляются в конец очереди, а продлеваемый путь удаляется. Если первый элемент очереди — это путь который ведет в конечную вершину, поиск можно завершить. Найденный путь гарантированно будет кратчайшим.

```
search_bridth(X,Y):-
    write('BFS start for': X), nl,
    get_time(BFS_START),
    bridth([[[]]],X,[],Y,L),
    inv_print(L),
    get_time(BFS_END),
    write('BFS END'), nl, nl,
    T1 is BFS_END - BFS_START,
    write('BFS time: '), write(T1), nl, nl.
```

Поиск с итеративным углублением использует идею метода поиска в глубину, однако глубина поиска ограничивается некоторым значением, поэтому необходимо ограничить длину возможных решений, что позволяет найти кратчайший путь.

```
search_id(Start,Finish) :-
    write('ITER start for': Start), nl,
    get_time(ITER_START),
    int(DepthLimit),
    depth_id([[]],Start,[],Finish,Res,DepthLimit),
    inv_print(Res),
    get_time(ITER_END),
    write('ITER END'), nl, nl,
    T1 is ITER_END - ITER_START,
    write('Iteratrion time: '), write(T1), nl, nl.
```

Для удобства используется предикат-обёртка solve()

```
solve(Start, Finish) :-
    search_deep(Start, Finish),
    search_bridth(Start, Finish),
        search_id(Start,Finish).
```

Результаты

?- solve([w, w, w, w, b, b, b, b],[w, b, w, b, w, b, w, b]).

DFS start for:[w, w, w, w, b, b, b, b]
[]
[b]
[w, b]
[b, w, b]
[w, b, w, b]
[b, w, b, w, b]
[w, b, w, b, w, b]
[b, w, b, w, b, w, b]
[w, b, w, b, w, b, w, b]
DFS END

DFS time: 0.35992002487182617

BFS start for:[w, w, w, w, b, b, b, b]
[]
[b]
[w, b]
[b, w, b]
[w, b, w, b]
[b, w, b, w, b]
[w, b, w, b, w, b]
[b, w, b, w, b, w, b]
[w, b, w, b, w, b, w, b]
BFS END

BFS time: 2.7340199947357178

ITER start for:[w, w, w, w, b, b, b, b]
[]
[b]
[w, b]
[b, w, b]
[w, b, w, b]
[b, w, b, w, b]

[w, b, w, b, w, b]
[b, w, b, w, b, w, b]
[w, b, w, b, w, b, w, b]
ITER END

Iteratrion time: 1.796919822692871

true

Как видно поиск в глубину завершился быстрее всего, но то и понятно, коль он ищет некратчайший путь.

Выводы

По ходу лабораторной работы изучен метод поиска в пространстве состояний для решения задачи. Реализованы три алгоритма поиска в графах: в ширину, глубину и с итеративным углублением.

Видно, что самым быстрый — DFS, но он не ищет кратчайший путь, а значит намного длиннее. BFS же его ищет, но медленее и попутно жадно потребляя оперативную память, поэтому лучше всего для такой задачи подходит поиск с итеративным углублением.