Developing Backtracking Algorithm to Find the Optimal Solution Path

Dr. Suhad M. Kadhum* & Isra'a A. Abdul-Jabbar*

Received on: 8/2/2010
Accepted on: 2/12/2010

Abstract

There are numerous search methods in A.I used to find the solution path to a subjected problem, but many of them return one solution path with no consider it is the optimal or not.

The aim of this work is to find a direct path from the start state to the goal state such that it is the shortest path with minimum cost (the optimal solution path).

We develop the backtracking algorithm in order to find the optimal solution path, such that all possible paths of the problem that expected to contain the optimal solution path can be checked, also we use a heuristic function depends on the actual cost of transition from one state to another. And in order to reduce the search time we discard any path that it is not useful in finding the optimal solution path.

The proposed algorithm was implemented using visual prolog 5.1 and tested on tree diagram and the result was good in finding the optimal solution path (with efficient search time equivalent to $O(b^{d/2})$ and space complexity O(bd) in worst cases).

Keywords: Artificial Intelligence, Search Algorithm, Optimal Path, Heuristic Search, Backtracking Strategy.

تطوير خوارزمية الرجوع لإيجاد المسار الامثل للحل

الخلاصة

هناك العديد من طرق البحث في الذكاء الاصطناعي المستخدمة لإيجاد مسار حل للمشكلة المطروحة، لكن العديد منها ترجع مسار حل واحد دون الاخذ بنظر الاعتبار هل ان هذا المسار يمثل الحل الامثل ام لا.

الهدف من هذا البحث هو ايجاد مسار مباشر من الحالة الابتدائية الى الحالة الهدف باقل كلفة وباقصر طريق(المسار الامثل للحل).

تم تطوير خوارزمية الرجوع للخلف لإيجاد المسار الأمثل للحل، بحيث ان كل المسارات الممكنة التي من المتوقع ان تحتوي على المسار الامثل للحل سوف يتم تدقيقها، وقد استخدمنا دالة موجهة تعتمد على الكلفة الحقيقية للانتقال من حالة الى اخرى. ولتقليل وقت البحث سنهمل أي مسار غير مفيد في ايجاد المسار الامثل للحل.تم تنفيذ الخوارزمية المقترحة باستخدام لغة برولوك المرئية 5.1 وتم اختبارها على مخطط شجري، وكانت النتائج جيدة في ايجاد المسار الامثل للحل (مع كفاءة في وقت البحث تقارب $O(b^{d/2})$ وتعقيد O(bd) في اسوء الحالات).

*Computer Science Department, University of Technology/ Baghdad

https://doi.org/10.30684/etj.28.24.13
2412-0758/University of Technology-Iraq, Baghdad, Iraq

1. Introduction

As far as search algorithm is concerned it is a global problem solving mechanism in artificial intelligence. Search algorithms are used for a multitude of AI tasks one of them is path finding .AI area of search is very much connected to problem solving. AI has investigated search methods that allow one to solve path problems in large domains. Having formulated problems we need to solve them and it is done by searching through the state space during this process search tree is generated by taking initial state and applying the successive function to it [1].

In computer science, a search algorithm is an algorithm that takes a problem as input and returns a solution to the problem, usually after evaluating a number of possible solutions. Most of the algorithms studied by computer scientists that solve problems are kinds of search algorithms. The set of all possible solutions to a problem is called the search space. Brute-force search or uninformed search algorithms use the simplest method of searching through the search space, whereas informed search algorithms use heuristic functions to apply knowledge about the structure of the search space to try to reduce the amount of time spent searching [2].

Blind searches will find any path. Heuristic searches will (usually) find any path, but will do so faster (usually) than blind search. Sometimes it's ok to find just any path to the goal as long as you get there. But sometimes you want to find the best path to the goal. The fastest, cheapest, or easiest route to take is oftentimes more important than finding some path. That's where optimal search comes in the methods that follow are intended to find the optimal path. One time-honored way of doing this is to find a method to measure the "goodness" of a state that is, to determine how close a given state is to the goal state. If we could make that evaluation consistently and correctly, then when we look at a list of states in trying to decide which to use next to generate new states, we could pick the state closest to the goal, instead of just picking the first one we see or picking one at random[3].

Our proposed search algorithm will not find any path to the goal only but tried to get all possible paths for the problem (except those that not useful in finding the optimal solution path) then decide the optimal path to the goal according to its cost and its number of states.

2. Background

Newell and Simon in 1976 defined that intelligent behaviors come from the manipulation of symbol entities that represent other entities and that process by which intelligence arises is called heuristic search [1].

2.1 Problem Solving and Search

A search algorithm takes a problem as input and returns the solution in the form of an action sequence. Once the solution is found, the actions it recommends can be carried out. This phase is called the execution phase. After formulating a goal and problem to solve, the agent calls a search procedure to solve it. A problem can be defined formally by four components which are [1]:

· The Initial State: The state in which agent starts.

· Successor function: Description of

possible actions and their outcomes.

· Goal Test: It determines that if the given state is the goal state.

· Path Cost: It is the summation of the actual cost of transition from one state to another.

2.2 Depth First Search

In depth – first – search, when a state is examined, all of its children and their descendants are examined before any of its siblings. Depth – first search goes deeper in to the search space when ever this is possible only when no further descendants of a state can found [4].

2.3 Backtracking [4]

Backtracking is a systematic method to iterate through all the possible states of a search space. Backtracking is really just depth-first search but it can support a direct solution path.

Backtracking can easily be used to iterate through all subsets or permutations of a set. Backtracking ensures correctness by enumerating all possibilities. For backtracking to beneficent, we must prune the search space.

Backtracking search begins at the start state and pursues a path until it reaches a goal or "dead end", if it reaches a goal, it returns the solution path and quits. If it reaches a dead end, it backtracks to the most recent node in the path having unexamined siblings and continues down on of those branches.

The algorithm of backtracking search is as follow:

```
{
SL:=[start];                NSL:=[start];
DE:=[];                     CS:=start;
While NSL!=[]
{
If CS=goal then Return SL;  /* success*/
If CS has no children (except on DE, SL, NSL) then
{
While SL!=[] and CS=first element of SL
{
Add CS to DE;  /* dead end*/
Remove first element of SL;
Remove first element of NSL;
CS:=first element of NSL;
}
Add CS to SL;
}
Place children of CS (except those on DE, SL, NSL) on NSL
CS:= first element of NSL;
Add CS to SL;
}
Return fail; /* failure*/
} /* end algorithm*/
```

2.4 Heuristics Search

Heuristic is a problem specific knowledge that decreases expected search efforts. It is a technique which sometimes work but not always. Heuristic search algorithms use information about the problem to help directing the path through the search space. These searches use some functions that estimate the cost from the current state to the goal presuming that such function is efficient. Generally heuristic incorporates domain knowledge to improve efficiency over blind search .In AI heuristic has a general meaning and also a more specialized technical meaning. Generally a term heuristic is used for any advice that is effective but is not guaranteed to work in every case.

2.4.1 Best First Search

Best first search is one of the most common heuristic search methods, and in this method, it uses an evaluation function and always chooses the next node to be that with the best score.

The basic algorithm of best first search is as follow [5]:

1. tart with open=[initial-state].
2. hile open !=[] do
   a. ick the best node on open.
   b. f it is the goal node then return with success. Otherwise find its successors.
   c. ssign the successor nodes a score using the evaluation function and add the scored nodes to open.

3. The Proposed Search Strategy

We develop the backtracking algorithm (described in 2.3) in order to find the optimal solution path, and we use a heuristic function depends on the actual cost of transition from one state to another.

In this method we will take the start state as the initial state and work forward chaining scanning for the goal by using the concept backtracking algorithm. And the cost of the current path is calculated by adding the cost of the current state to the previous cost each time we reach a new state, and if a dead end state (the state with no children) appears, the search return up trying to find another solution path (backtracking) and discard the cost of all dead states. If the parent state has no other child (except those dead ends) then we will deal this state as a dead end also and continue this steps until we find the first goal (if it's exist) the solution path for this goal will be a direct path from the start state to the end state. This algorithm will continue trying to find all possible solution paths in order to decide the optimal one. We will deal the goal state as a dead end after storing its path and its cost in a temporary variable in the external database. If another solution path found then its cost will be compared with the stored one to decide which one is the best, also we take the number of states of the solution paths in our account. And each time we reach any path that have a cost higher than (or equal to) the stored cost, we will discard that path and dealing with the current state as a dead end, until all the tree of the problem was searched, then the stored path will be the optimal solution path and its cost will be the optimal one.

3.1 Data Representation

The problem was represented as a tree and the data of this tree was represented in the program as logical terms, where each term has he following form:

move (State1, State2, Cost).

Where:

move: the predicate name.

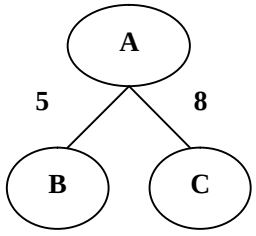
State1: the parent node and has a "symbol" data type.

State2: the child node and has a "symbol" data type.

Cost: the actual cost of transition from state1 to state2 and has an integer data type.

These logical terms will represent the input to the proposed algorithm.

Such as if we have the following sub tree:



The data of this sub tree was represented in the program as:
move (A,B, 5).
move (A,C, 8).

3.2 The Design of the Proposed Algorithm

The proposed algorithm uses three lists, they are:

1. **NSL**: The new state list contains nodes waiting evaluation, nodes whose descendants have not yet been generated and searched. And this is a list of the logical terms:

n(State, C)

Where:
n: is the predicate name.
State: symbol represents the state.
C: integer value represents the cost of this state.

2. **SL:** The state list, lists the states in the current path being tried, if a goal is found, SL contains the ordered list of states on the solution path (direct path from the start state to the goal state) , and this is a list of logical terms:

s (State, C, L)

Where:
s: the predicate name.
State: symbol represents the state.
C : integer value represents the cost of this state.
L : list of symbols represents the children of this state.

3.
E: dead end it is a list of symbols, the state is put in this list in one of the following cases:
-The state that have no child.
-The state that all its descendants are found in DE and not found in NSL.
-The state that it is a goal.
-The state that we expect it is not useful to find optimal path (when the cost of the current path is greater or equal to the stored cost of current optimal solution path).

3.3 The Proposed Search Algorithm

Input: logical terms represent the tree of the problem.
Output: A list of states represents the direct optimal solution path(P), and its cost (C).

Process:

```
{
  P= []; /* the optimal path*/
 C=LARGE_VALUE;
/*The cost of the optimal path*/
    CC= 0;
/* the cost of the current path*/
   NSL:= [n (Start, 0)];
   SL:= [];
   DE:= [];

Step1: If (NSL ==[]) then goto  Step5;
 Remove the first term (n(X, N)) from
NSL; /* CS */
      CC =N+CC;
      If(X is the goal state) then
      {
        List1= [];
/* discard all its children if any *
 Let Len1 be the length of SL;
Let Len2 be the length of P;
If ((CC <C) or (CC == C and Len1<
Len2)) then
 {
  Let P be the list of states of SL;
  C= CC;
 }
      Goto Step2 ;                                    D
}
     If (CC >= C) then
 /* discard this path*/
{
        List1:=[];
        Goto  Step2 ;
 }
   Get all children of X with their
     cost and put them in List2;
 /*  as  logical  terms  with  predicate
name n */
```

```
    Let List1 be a list of states of List2;
Step2: Add the term (s(X, N, List1)) to the SL;
/*to the beginning of SL */
 If (List1! = []) then
   {
  Add List2 to the NSL; /* to the beginning of the NSL*/
 Goto Step1;
 }
Step3: Let s(H, K, CL) be the first term in SL;
Step4: If (CL== []) then
       {
         CC = CC -K;
      remove s(H,K,CL)from SL;
         Put H in DE;
         Goto Step3;
       }
       Let S1 be the first element in CL;
       If ((S1 is member in DE)
             AND
(S1 is not member in the states of NSL)) Then
        {
          Remove S1 from CL;
          Goto Step
        }
       Goto  Step1;
Step5: If (P== []) then Return failure;
 /* the goal is not found*/
        Print the optimal path (P) and its minimum cost (C);
  }
/*end the algorithm*/
```

3.4 Implementation of the Proposed Algorithm

The Proposed algorithm implemented with visual prolog and tested by tree diagram shown in figure (1), which has start state represented by the node (A) and the goal state represented by the node (K), this graph shown that there are six nodes in the diagram known as (K), the proposed strategy returned the direct optimal solution path from (A to K) with minim cost and shortest path, the implementation result shown in Table (1).

4. Discussion

Sometimes we want to find just any path to the goal (solution path), but sometimes we want to find the best path to the goal (optimal solution path) and this is the aim of our work.

In this paper we suggest a method that find all possible solution paths (that we expect it may be the optimal) and then take the optimal one.

At the first step we use the depth-first search in order to find the first solution path but we find that this method not support us with the direct path to the goal and it is also test all states, which it is in sometimes considered to be time consuming, therefore we develop our proposed search by using the facility of backtracking search by discarding the states that not lead to direct solution path. Also we used the concept of heuristic search that support each transition from a state to another with a cost that useful in finding the optimal solution path.

After we find the first direct solution path, we store this path and its cost, then we develop the proposed algorithm in order to find all possible solution paths, and each time we find a new solution path we compare its cost and number of its states with the stored one in order to find the optimal path. And in order to reduce search time we develop the proposed algorithm by adding some conditions and constraints that prevent testing states and paths that it is not useful in finding the optimal solution path.

We compare the proposed

algorithm with one common heuristic search method (described in 2.4.1), and found it is a good method in finding the direct optimal solution path with efficient search time, see the table(1).

To discuss the result of the proposed search a complete comparison was produced between our method and other search method like best first search and A* algorithm in worst time and space complexity in the execution of algorithms see table (3).

If each node has b descendants, then
Level0 (the root node) has 1 node
Level 1 has b node
Level 2 has b*b node
Level 3 has $b^2*b=b^3$
.
.
Level d has $b^{d-1}*b = b^d$

If a decent heuristic for ordering moves can be found , then half the nodes need not to be evaluated, therefore the time complexity is cut in half and be $O(b^{d/2})$.The space is dominated by the size of the queue that have list of partial paths and in worst case is equivalent to the complexity of depth-first search depending on the goal on leaf node of the tree and this require (d) times ,until reach to leaf node.

5. Conclusions

In this paper the following points can be concluded:

1 Developing the backtracking algorithm and using the concept of heuristic search can be consider as a good method in finding the optimal direct solution path.

2 In order to find the optimal solution path we must search all possible solution paths that we expect it may contain the optimal solution path.

3 Getting an efficient search time by discarding the paths that we expect to be not useful in finding the optimal solution path.

4 Checking all the tree's states of the problems consider to be time consuming, so we develop the search strategy by adding some conditions and constrains that lead the search direction such as:

- If the state is the goal then we will discard all its descendent because it will not contain solution path better than the found one.
- Each time we reach a new state, such that the cost of the path under test become larger than or equal to the stored one we discard all the descendent of this state.

References:

[1]Kamran Zaheer ,"Artificial Intelligence Search Algorithms In Travel Planning", Department of Computer sciences and Electronics Mälardalen University Västerås Sweden, 2006.

[2]Poli, Langdon, and McPhee," A Field Guide to Genetic Programming", 2009.

[3]Senior Diablo," Beginners Guide to Path finding Algorithms", 2007.

[4]William A. Stubblefield & Luger E.George, "Artificial Intelligence and the Design of Expert Systems", 1998.

[5]Stuart J. Russell and Peter Norvig," Artificial Intelligence: A Modern Approach Prentice Hal", 2003.

Table (1): The Proposed Algorithm Implementation Result

"The optimal solution path is [A-D-K] and its Cost is 6"

I	CS	SL	NSL	DE	CC	C
1	n(A,0)	[]	[n(A,0)]	[ ]		
2	n(B,3)	[s(A,0,[B, C,D,P])]	[n(B,3),n(C,2),n(D,1),n(P,4)]	[ ]	0	
3	n(E,4)	[s(B,3,[E,F]), s(A,0,[B, C,D,P])]	[n(E,4),n(F,5),n(C,2),n(D,1),n(P,4)]	[ ]	3	
4	n(F,5)	[s(B,3,[E,F]), s(A,0,[B, C,D,P])]	[n(F,5),n(C,2),n(D,1),n(P,4)]	[E]	8	
5	n(K,3)	[s(F,3,[K]), s(B,3,[F]), s(A,0,[B, C,D,P])]	[n(K,3),n(C,2),n(D,1),n(P,4)]	[E]	11	11
6	n(C,2)	[s(A,0,[C,D,P])]	[n(C,2),n(D,1),n(P,4)]	[BFKE]	2	
7	n(G,1)	[s(C,2,[G,H]), s(A,0,[C,D,P])]	[n(G,1),n(H,2),n(D,1),n(P,4)]	[BFKE]	3	
8	n(L,2)	[s(G,1,[L]), s(C,2,[G,H]), s(A,0,[C,D,P])]	[n(L,2),n(H,2),n(D,1),n(P,4)]	[BFKE]	5	
9	n(H,2)	[s(C,2,[H]), s(A,0,[C,D,P])	[n(H,2),n(D,1),n(P,4)]	[GLBFKE]	3	
10	n(D,1)	[s(A,0,[D,P])]	[n(D,1),n(P,4)]	[CHGLBFKE]	1	
11	n(I,4)	[s(D,1,[I,K]), s(A,0,[D,P])]	[n(I,4),n(K,5),n(P,4)]	[CHGLBFKE]	5	
12	n(M,4)	[s(I,4,[M]), s(D,1,[K]), s(A,0,[D,P])]	[n(M,4),n(K,5),n(P,4)]	[CHGLBFKE]	9	
13	n(K,1)	[s(M,4,[K]), s(I,4,[M]), s(D,1,[K]), s(A,0,[D,P])]	[n(K,1),n(K,5),n(P,4)]	[CHGLBFKE]	10	10
14	n(K,5)	[s(D,1,[K]), s(A,0,[D,P])]	[n(K,5),n(P,4)]	[IMKCHGLBFKE]	6	6
15	n(P,4)	[s(A,0,[P])]	[n(P,4)]	[DKIMKCHGLBFKE]	1	
16	n(Q,1)	[s(P,4,[Q,K]), s(A,0,[P])]	[n(Q,1),n(K,7)]	[DKIMKCHGLBFKE]	4	
17	n(R,5)	[s(Q,1,[R]), s(P,4,[Q,K]), s(A,0,[P])]	[n(R,5),n(K,7)]	[DKIMKCHGLBFKE]	5	
18	n(K,7)	[s(P,4,[K]), s(A,0,[P])]	[ ]	[QRDKIMKCHGLBFKE]	11	

CS: The current state (state under test).
SL, NSL, and DE: is a state list, new state list and dead end list respectively.
CC: is the current cost of the path under test.
C: is the cost of optimal solution path.

Table (2): Comparison between the proposed method and Best –First search

The Proposed Method	Best First Search
Find the optimal solution path, since all the tree of the problem was searched.	Find the first best solution path with no consider it is the optimal or not.
Efficient search time because of discarding the paths that not contain the optimal path.	There are no additional conditions or constraints to reduce the search time.
Take the number of states of optimal path in its account.	Does not take the number of states of optimal path in its account.
Find a direct solution path (from the start state to the goal states).	Cannot find a direct solution path, since each time, the states rearranged according to its cost.

Table (3): Comparison among the proposed method and other search strategies in both time and complexity

Search method	Time	Space
Depth- first	$O(b^{d+1})$	O(b d)
Best-first	$O(b^{d+1})$	$O(b^{d})$
A*	$O(b^{d+1})$	$O(b^{d})$
The Proposed method	$O(b^{d/2})$	$O(b^{d})$

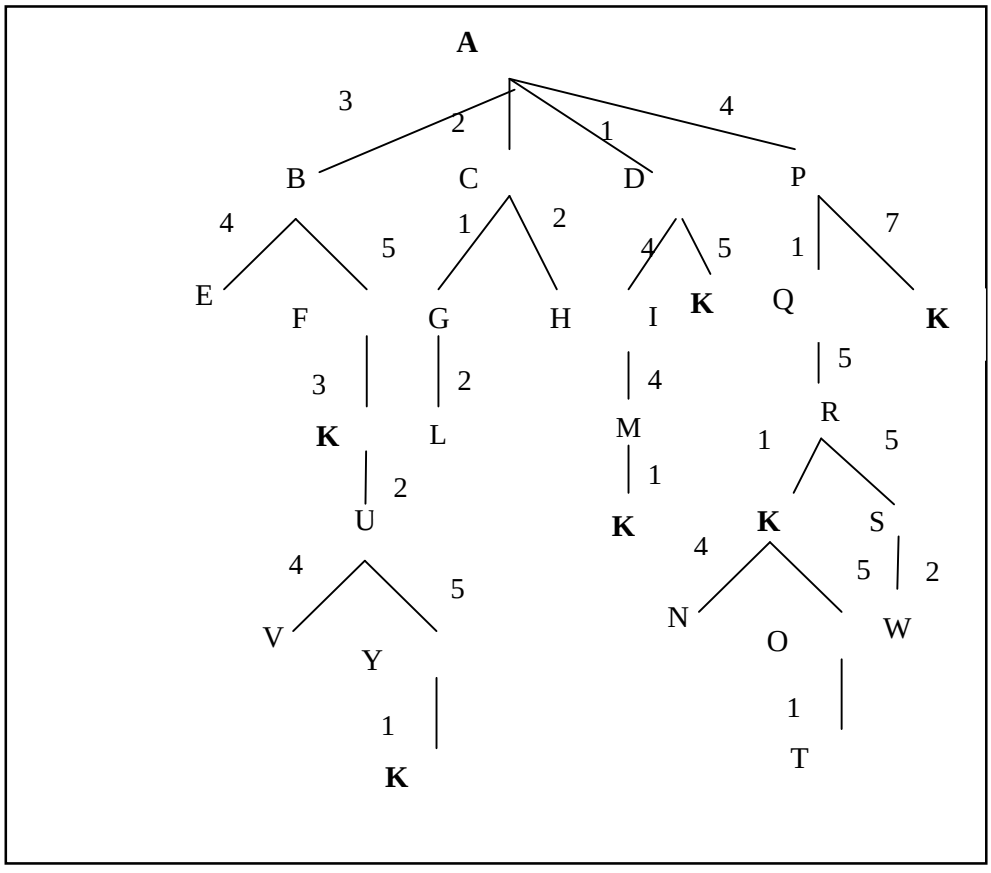


Figure (1): Tree Graph Example