Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol.5, No.2, 2014

Adaptive Backtracking Search Strategy to Find Optimal Path for Artificial Intelligence Purposes

Isra'a Abdul-Ameer Abdul-Jabbar[1, 2*] Suhad M. Kadhum[2]

1. School of Computer and Information, Hefei University of Technology, Hefei 230009, People's Republic of China
2. Computer Science Department, University of Technology, Baghdad, Iraq

* E-mail of the corresponding author: israa_ameer@yahoo.com

Abstract

There are numerous of Artificial Intelligence (AI) search strategies that used for finding the solution path to a specific problem, but many of them produce one solution path with no attention if it is the optimal path or not. The aim of our work is to achieve the optimality by finding direct path from the start node to the goal node such that it is the shortest path with minimum cost .In this paper adaptive backtracking algorithm is produced to find the optimal solution path, such that all possible paths in the tree graph of the search problem that have an expected optimal solution is tested, also a heuristic function related to the actual cost of the moving from one node to another is used in order to reduce the search computation time. The adaptive algorithm ignored any path that it is not useful in finding the optimal solution path, our adaptive algorithm implemented using visual prolog 5.1, evaluated on tree diagram and produced good result in finding the optimal solution path with efficient search time equivalent to $O(b^{d/2})$ and space complexity $O(b^d)$.

Keywords: Backtracking Algorithm, Optimal solution Path, Heuristic function, Dead end, shortest path, Minimum cost.

1. Introduction

In so far as the search algorithm is related to a global problem solving mechanism in artificial intelligence, the search algorithms are used for a multitude of artificial intelligence tasks as one of them, since finding a solution path by search algorithms in artificial intelligence area is very connected to problem solving. Artificial intelligence investigated a search methods that allow one to solve path problems in large domain spaces, this is can be done by looking through the state space of the search tree that generated by initialized the start node and applying the consecutive process to the next nodes (Zaheer K., 2006).

In AI, a search algorithm is a procedure steps that select a problem as input and gives a solution to the problem as output, usually after testing a number of all possible solutions. Many of the algorithms proposed by scientist researches that solve problems are kinds of search algorithms. The collection of all possible solutions to a problem is called the search space. Brute-force search or uninformed search algorithms is the simplest method of searching through the search space, since the informed search algorithms used heuristic functions to draw the structure of the search space as trying to minimize both of the solution cost and the amount of the time spent in searching (Poli R., et al. , 2009).

Artificial intelligence search strategies divided to two types: blind searches and heuristic searches. Blind searches try to find any path and the heuristic searches also usually try to find any path, but usually work so faster than blind search. Sometimes it's good to find just any path to the goal as long as you get there, But for best solution sometimes we need to to find the best path to the goal, the best path means we try to extract the cheapest, fastest, or the easiest route to take is oftentimes more important than finding some paths. That's where optimal search comes in the methods that are intended to find the optimal path. Many of previous ways that can do this is try to find a good measurement method to estimate the goodness of a specific node and to detect how close a given node is to the goal node. If we could make that evaluation correctly and consistently , then when we look at a set of states in trying to decide which to use next to generate new node states, we could select the state closest to the goal, instead of just taking the first node we found or selecting random one (Diablo S., 2007).

Our adaptive backtracking search algorithm will not find any path or route to the goal only, but it is also try to get all possible paths for the problem except those that not useful in finding the optimal solution path then picks the optimal path to the goal according to its cost and the number of node states (path length).

Section 2 gives background to common search strategies, section 3 described the proposed search strategy and the design of the adaptive backtracking search, section 4 described the experiment and result, section 5 contains the discussion and finally , the conclusions are explained in section 6.

2. Background

In 1976, Newell and Simon found that the intelligent behaviors come from the doctrinaire of model entities that represent other entities and that process by intelligence arises is known as heuristic search.

2.1 Problem Solving and Search

The search algorithm takes a problem as tree graph input and returns the solution in the form of an action process. Once the solution is found, the actions it recommends can be implemented. This stage is called the execution stage. After educing the goal and the problem to solve, then the search procedure is called to solve it. The problem can be determined formally by four elements which are (Zaheer K., 2006):

• The start state:

• Procedure or function: Displayed the possible actions and their outputs.

• Goal Test: determines if the given state is the goal state or not.

• Path Cost: It is the summation of the actual cost of moving from one node state to another.

2.2 Depth First Search

In depth first search, when a node state is examined, all of its seed and their descendants are tested before any of its siblings. Depth first search works deeper in to the search space when ever this is possible only when no further descendants of a state can found (Stubblefield W. A. and George F. L, 1998).

2.3 Backtracking

Backtracking (Stubblefield W. A. and George F. L, 1998) is a formal method to get back through all the possible states of a search space. Backtracking in fact is really just depth first search, but it can produce a direct solution path. Backtracking can simply be used to return through all subsets or permutations of a set. Backtracking guarantees correctness by enumerating all possibilities. For backtracking to beneficent, we must cut the search space.

Backtracking search begins at the start state node and keep a track a path until it reaches a goal state node or "dead end", if it arrives a goal, it gives the solution path and quits. If it arrives a dead end, it backtracks to the unprecedented node in the path having untested siblings and goes on down 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 search is a function with a specific knowledge that reduced the expected search efforts. It is a technique which in some times produces best path, but not always gives the optimal path. Heuristic search algorithms employ information about the problem to assist in finding the direct path through the search space. These searches utilize some functions that evaluate the cost from the current node state to the goal node state presuming that such search is efficient. Generally heuristic shaped the domain knowledge to evolve efficiency over blind search .In Artificial intelligence; heuristic has a public meaning and also a more expert technical meaning. Generally, heuristic expression is usually used for any counsel that is efficient but is not ensured to work in every case.

2.4.1 Best First Search

Best first search (Russell S.J and Peter N., 2003) is one of the most known heuristic search algorithms; this method uses an evaluation function and always picks the next node to be that with the best cost.

The basic algorithm of best first search is as follow:

1. Start with open=[start state].
2. While open!=[] do
3. Pick the best node on open.
4. If it is the goal node then return with success. Otherwise find its successors.
5. Specify the successor nodes a cost (score) using the evaluation function and add the score nodes to open.

3. The Proposed Search Strategy

In this section , We produced the adaptive backtracking algorithm to one described in section 2.3 in order to find the optimal solution path, and a heuristic function relied on the actual cost of moving from one state to another is used.

In this strategy we will take the start node state as the initial state and work in a forward chaining searching for the goal by using the concept of backtracking algorithm. And the cost of the current path is computed by adding the cost of the current state to the previous cost each time we arrive a new state, and if a dead end state (the state with no children) appears, the search back up trying to find another solution path (backtracking) and discard the cost of all dead states. If the current root (parent) state has no other child (except those dead ends) then we will treat with this state as a dead end also and go on this steps until we find the first goal (if available) the solution path will be a direct path from the start node to the end node. This algorithm will continue to find all possible solution routes in order to pick the optimal one. We will deal the goal state as a dead end after keeping its path and its cost in a temporary stack in external database. If the search found another solution path, its cost will be compared with the one stored in the database to decide which one is the best, also we take the number of nodes of the solution paths in our account. And each time we arrive 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 state space of problem was represented as a tree graph 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:

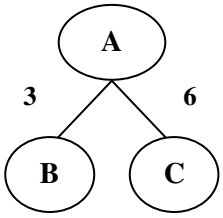
move: The predicate name.

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, 3), move (A, C, 6).



3.2 The Design of the Adaptive backtracking Algorithm

The proposed algorithm uses three lists, they are:

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.

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.

DE: 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 Adaptive 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:= [];
XX: If (NSL ==[]) then goto XX4;
    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 XX1 ; }
If (CC >= C) then /* discard this path*/
 { List1:=[];
  Goto XX1 ;
 }
 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;
XX1: 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 XX; }
XX2: Let s(H, K, CL) be the first term in SL;
XX3: If (CL== []) then
 { CC = CC -K;
       Remove s(H,K,CL)from SL;
   Put H in DE;
   Goto XX2;}
     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 XX3; }
     Goto XX;
XX4: If (P== []) then Return failure; /* the goal is not found*/
      Print the optimal path (P) and its minimum cost (C);
  } /*end the algorithm*/
```

4. Experiment and Result

The Proposed algorithm implemented with visual prolog 5.1 and tested by tree diagram of problem shown in figure (1), which has start node state represented by the root node (A) and the goal node state represented by the node (K), this graph illustrated that there are six nodes in the diagram known as (K), the adaptive strategy returned the direct optimal solution path from (A to K) with shortest path and minim cost, the result is illustrated in table (1).

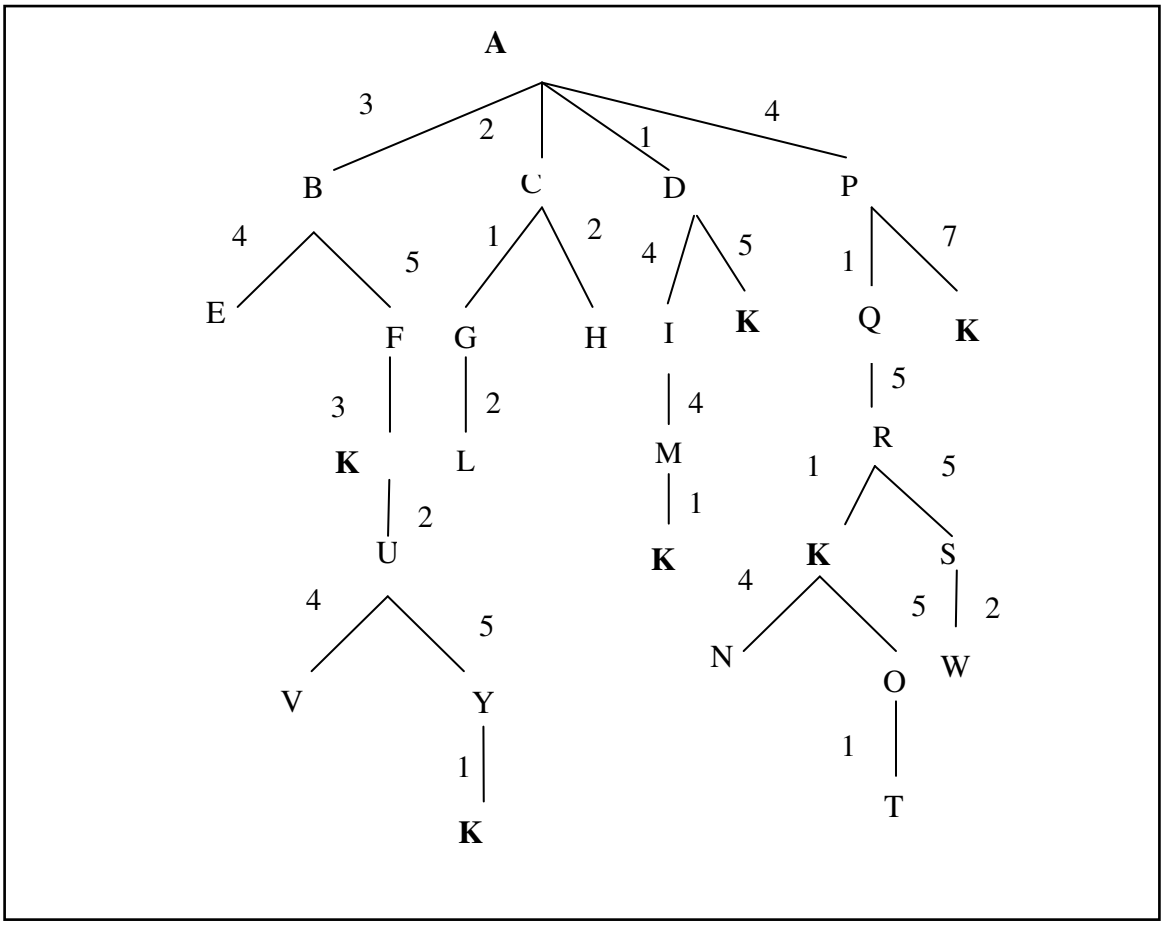


Figure 1. Tree graph example

Table 1. The Proposed Algorithm Implementation Result

CS: The current state (state under test). SL, NSL, and DE: is a state list, new state list and dead end list

CC: Is the current cost of the path under test. C: Is the cost of optimal solution path.

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]),	[n(R,5),n(K,7)]	[DKIMKCHGLBFKE]	5	

		s(P,4,[Q,K]), s(A,0,[P])]				
18	n(K,7)	[s(P,4,[K]), s(A,0,[P])]	[ ]	[QRDKIMKCHGLBFKE]	11	

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

5. 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 produces an adaptive strategy 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 the finding of the first direct solution path, we store this path with its related cost, then the our adaptive algorithm is activated in order to find all possible solution paths, and each time we find a new solution path we compare its cost and the 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 compared 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, as illustrated in table (2).

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 as illustrated in table (3).

If each node has b descendants, this mean the Level 0 (the root node) has 1 node, Level 1 has b node, Level 2 has b*b node, Level 3 has b^2*b=b3, Level d has b^{d-1}*b =bd.

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.

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

The Adaptive Method	Best First Search
Find the optimal solution path, since all the tree of the problem was scanned.	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 node states of optimal route in its account.	Does not take the number of node states of optimal route in its account.
Find a direct solution path from the start node to the goal node.	Cannot find a direct solution path, since each time, the states sorted according to its cost.

Table 3. Comparison among the proposed method and other search strategies in both time and complexity

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

6. Conclusion

In this paper, an adaptive backtracking search algorithm is produced using the concept of heuristic search. Our search strategy can achieved a good result in finding the direct - optimal solution path. Since in order to find the optimal solution path the adaptive algorithm must search all possible solution paths that may contain the optimal solution path; also the adaptive strategy produced an efficient performance in term of search time by discarding the paths that expecting to be not useful in finding the optimal solution path. Checking all the tree's node states of the search problem consider to be time consuming, so we developed the search strategy by adding some conditions and constrains that guide the search direction such as:

-If the state is the goal, we will discard all its descendent because it will not contain a solution path which is better than the one is found.

- Each time a new state node is reached, such that if the cost of the tested path becomes larger than or equal to the stored one, we will discard all the descendent of this node.

Our adaptive approach produced two advantages that required in achieving the best performance in heuristic search strategy, in both terms of reducing the time complexity and finding the optimal solution route with the shortest path and the minimized cost.

References

Diablo, S. (2007). Beginners Guide to Path finding Algorithms. http:// ai-depot. com/Tutorial/ PathFinding. html .

Poli, R., Langdon, W. B., McPhee, N. F. (2009). A Field Guide to Genetic Programming. http:// www. e-booksdirectory. com/details.php?ebook=2275

Russell S.J and Peter N. (2003). Artificial Intelligence: A Modern Approach .Prentice Hal 3rd edition. Englewood Cliffs, New Jersey 0763.

Stubblefield W. A. and George F. L.(1998). Artificial Intelligence and the Design of Expert Systems. Benjamin-Cummings Publishing Co., Inc. Redwood City, CA, USA.

Zaheer, K. (2006).Artificial Intelligence Search Algorithms In Travel Planning. Department of Computer sciences and Electronics Mälardalen University Västerås Sweden