STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIX**, Special Issue 1, 2014
10th Joint Conference on Mathematics and Computer Science, Cluj-Napoca, May 21-25, 2014

SOME IMPROVEMENTS OF THE EXTENDED BREADTH-FIRST SEARCH ALGORITHM

TAMÁS KÁDEK AND JÁNOS PÁNOVICS

Abstract. Extended breadth-first search (EBFS) is an algorithm developed to give remedy to some problems related to the classical state-space representation used in artificial intelligence. This algorithm was initially intended to give us the ability to handle huge state spaces. The authors have shown a number of examples of the practical use of EBFS since it was developed. Based on their experiences, they found some ways for improving the algorithm. This paper presents the new algorithm, which contains these improvements.

1. Introduction

The most basic problem representation technique used by artificial intelligence is state-space representation. However, it can be used to describe only a certain small subset of problems conveniently. For example, even a simple chess puzzle may have too many ways to continue from a particular situation because each piece can move in quite a few directions and potentially more than one square. Of course, we do not have to deal with all the possible moves in a particular situation, provided we have a means to mark the cases that are relevant regarding the solution's viewpoint. This can only be done only if we have some additional knowledge about the problem, which is usually represented by a heuristic function. In case we have this additional knowledge, however, it may still be difficult to describe it as a function. For example, in the 8-puzzle game, the heuristic function evaluates a situation as being more appealing if it has more pieces on their correct places, but this measure will fail in some cases. To handle this problem, we introduced the extended breadth-first search algorithm in [1].

Received by the editors: May 1, 2014.

2010 *Mathematics Subject Classification.* 68T20.

1998 *CR Categories and Descriptors.* I.2.8 [**Computing Methodologies**]: ARTIFICIAL INTELLIGENCE – *Problem Solving, Control Methods, and Search*.

Key words and phrases. artificial intelligence, state-space representation, extended model, breadth-first search.

2. The Extended State-Space Model (ESSM)

The EBFS algorithm can be defined after introducing an extended state-space model [2], which allows us to discover the representation graph starting from several different states and possibly in more than one direction. Using state-space representation, solutions to problems are obtained by executing a series of well-defined steps. During the execution of each step, newer and newer states are created, which form the state space. States are distinguished from one another based on their relevant properties. Relevant properties are defined by the sets of their possible values, so a state can be represented as an element of the Cartesian product of these sets. Let us denote this Cartesian product by S. Possible steps are then operations on the elements of S. Let us denote the set of operations by F. The state space is often illustrated as a graph, in which nodes represent states, and edges represent operations. This way, searching for a solution to a problem can be done actually using a path-finding algorithm.

We keep the basic idea (i.e., the concepts of states and operations on states) also in the extended state-space model (ESSM). The goal of this generalization is to provide the ability to model as many systems not conforming to the classical interpretation as possible in a uniform manner.

A state-space representation over state space S is defined as a 5-tuple of the form

$$\langle K, \text{initial}, \text{goal}, F, B \rangle,$$

where

- K is a set of initially known (IK) states, such that $K \subseteq S$ and $K \neq \emptyset$,
- $\text{initial} \in \{\text{true}, \text{false}\}^S$ is a Boolean function that selects the initial states,
- $\text{goal} \in \{\text{true}, \text{false}\}^S$ is a Boolean function that selects the goal states,
- $F = \{f_1, f_2, \ldots, f_n\}$ is a set of "forward" functions, $f_i \in (2^S)^S$,
- $B = \{b_1, b_2, \ldots, b_m\}$ is a set of "backward" functions, $b_i \in (2^S)^S$.

The "forward" and "backward" functions represent the direct connections between states. For more details, see [1].

Some notes:

- The number of initial and goal states is not necessarily known initially, as we may not be able to or may not intend to generate the whole set S before or during the search.
- The $n + m = 0$ case is excluded because in that case, nothing would represent the relationship between the states.
- Although the elements of the sets F and B are formally similar functions, their semantics are quite different. The real set-valued functions

in F are used to represent nondeterministic operators, while there may be real set-valued functions in set B even in case of deterministic operators.

Let us now introduce a couple of concepts:

- *Initial state*: a state s for which $s \in S$ and initial(s) = true.
- *Goal state*: a state s for which $s \in S$ and goal(s) = true.
- *Known initial state*: an initial state in K.
- *Known goal state*: a goal state in K.
- *Edge*: an $\langle s, s', o\rangle \in S \times S \times (F \cup B)$ triple where if $o \in F$, then $s' \in o(s)$, and if $o \in B$, then $s \in o(s')$.
- *Path*: an ordered sequence of edges in the form

$$\langle s_1, s_2, o_1\rangle, \langle s_2, s_3, o_2\rangle, \ldots, \langle s_{k-1}, s_k, o_{k-1}\rangle,$$

where $k \geq 2$.

General objective: determine a path from s_0 to s^*, where s_0 is an initial state, and s^* is a goal state.

3. The EBFS Algorithm

The EBFS algorithm extends the BFS algorithm with the ability to run more than one breadth-first search starting from more than one state (the initially known states). It is particularly useful if the subtrees explored reach one another as illustrated by Fig. 1. The dashed line denotes the subtree that is discovered by the standard BFS algorithm starting from i_1 if the nearest goal state is g_1. However, in case we give also the states k_1, k_2, k_3 besides i_1 as potentially useful states, then the discovered part of the graph is smaller, even if k_1 did not prove to be useful for finding the solution as the illustration shows.

The EBFS algorithm stores a subgraph of the representation graph during the search. The main difference from BFS at this point is that in case of EBFS, the relationship between the nodes and each IK state is stored.

The full pseudocode of the EBFS algorithm can be found in [1]. The database of the algorithm stores for each node the state represented by the node as usual, the forward and backward status (open, closed, or not relevant), forward and backward parents, forward and backward children of the node, as well as the distance from and to each of the IK states.

The main algorithm begins with an initialization step, which adds all IK states to the database as open states in both directions, i.e., they are all waiting to be expanded. Like BFS, the algorithm continuously expands a state in one of the directions until the goal condition is satisfied, or there are no more open states left. Checking the goal condition means that we have to

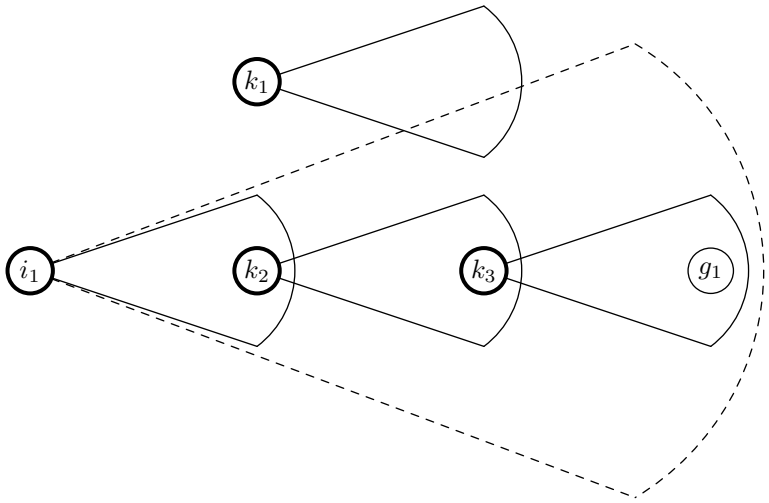

FIGURE 1. Subtrees reaching one another. [1]

```
procedure EBFS(K)
begin
   nodes ← INITIALIZE(K)
   while true do
      if { n | n ∈ nodes ∧ STATUS[n] = open } = ∅ then
         terminate unsuccessfully
      end if
      curr ← SELECT(nodes)
      EXPAND(curr, nodes)
      if GOAL-CONDITION(nodes) then
         terminate successfully
      end if
   end while
end procedure
```

FIGURE 2. The pseudocode of the controller.

determine whether there is an initial state s_0 and a goal state s^* such that s^* can be reached from s_0 via an initially known state. Similarly to the BFS algorithm, the state selected for expansion is the one with the smallest depth level. Remember that each state has as many depth levels as many IK states we have in the database, i.e., these depth levels are the distances to and from each IK state.

During expansion, we apply all the operators as usual and update the information in the database about the expanded state and the generated state. A recursive update of the stored information about the nodes is also required whenever an initially known state becomes reachable from another one during the search.

As you can see from the algorithm description, the initially known states play a very significant role in the algorithm. They form a point of reference among all the states explored during the search, as the relationship between each state and these IK states are stored in the database: for each state s (including the IK states themselves), we store the minimum number of steps in which each IK state can be reached from s in the already explored part of the graph, and the minimum number of steps in which s can be reached from each IK state. This way, some properties of the already explored part of the graph can be mapped to the IK states. For example, in the original algorithm, the termination condition only requires the IK states and their relationships with the initial and goal states, and not the whole database. There is only one limitation: at least one of the IK states should be on a path representing a solution. Whenever an initial state is included in K, this condition is satisfied.

4. Improving the EBFS Algorithm

To show the benefits of the EBFS, earlier we showed a state-space representation that suffers from the presented problems, so the state space is big enough. It was a representation for the well-known n-queens problem. In this representation, a state is defined by an $n \times n$ Boolean matrix, the cells of which represent the squares of a chessboard. An element of the matrix is true if there is a queen on that square and false if it is empty. We had as many operators as many squares on the chessboard. We know that this representation is far from the best choice, we only chose this because it has the drawbacks described earlier.

Later, we tried to get better results and wanted to support the ESSM model as much as possible, so we also enabled to use forward and backward functions together. Unfortunately, we were unable to use the benefits of having both forward and backward operators. The original EBFS algorithm works well if we use it on a state space that contains only forward functions. In the case of the original algorithm, we can only recognize the connection between two initially known states if the path between them was discovered in one direction only, i.e., we only used either forward or backward functions during the exploration. As you can see in Fig. 3, we could have recognized the connection between k_2 and k_3 when the first state s is reached that is accessible both from k_2 using forward functions and from k_3 using backward functions.

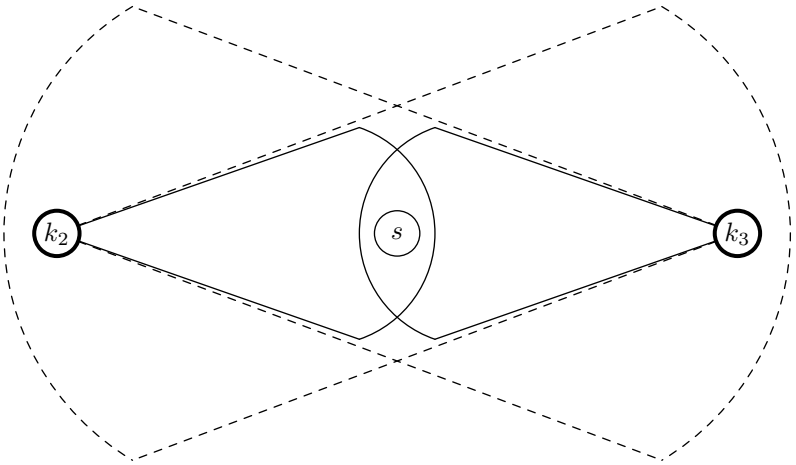

FIGURE 3. Subtrees reaching one another using the improved algorithm.

Comparing the total area covered by the two dashed lines in the figure with the total area covered by the two solid lines, we find that we have to explore much less states using the improved algorithm. Note that the areas try to show the radii of the subtrees explored starting from each of the initially known states.

4.1. **Using an Extended Neighborhood Matrix.** To improve our algorithm, the controller will maintain an extended neighborhood matrix and two vectors storing the accessibility of the initial and goal states from the initially known states:

$$\begin{pmatrix} 0 & q_{1,2} & \dots & q_{1,n} \\ q_{2,1} & 0 & \dots & q_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n,1} & q_{n,2} & \dots & 0 \end{pmatrix} \quad \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} \quad \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}$$

$q_{i,j}$ represents the number of function applications required to reach k_j from k_i, where $k_i, k_j \in K$. When a state like s is reached, which is accessible from k_i using n forward operators and accessible from k_j using m backward operators, then $q_{i,j}$ is set to $n + m$ if $q_{i,j} > n + m$. Initially, $q_{i,j}$ is set to 0 if $i = j$ and ∞ otherwise. Note that we also need to store s, the state providing the best connection between k_i and k_j, so that later we can recover the solution from the database. Consider the following example:

$$\begin{pmatrix} 0 & 5 & \infty & \infty \\ \infty & 0 & 4 & \infty \\ \infty & \infty & 0 & \infty \\ 3 & \infty & \infty & 0 \end{pmatrix} \quad \begin{pmatrix} 1 \\ \infty \\ \infty \\ \infty \end{pmatrix} \quad \begin{pmatrix} \infty \\ \infty \\ 2 \\ \infty \end{pmatrix}$$

Here, we have a solution of length 12, including k_1, which is accessible from an initial state using one operator, k_2, which is accessible from k_1 using five operators, k_3, which is accessible from k_2 using four operators, and a goal state, which is accessible from k_3 using two operators. Remember that the algorithm can stop in two different ways:

- If we have no more open nodes, the algorithm terminates without finding a solution. In this case, the problem is said to be unsolvable if the initial state was also initially known.
- If we find a solution, the algorithm terminates successfully, and the solution can be recovered based on the information stored in the database. To recognize that a solution is found, we have to run a search algorithm on the extended neighborhood matrix. We can consider this search as a search on a "metagraph." Note that this search increases the time complexity of the algorithm, but it is necessary only if the matrix has changed after the last expansion.

4.2. **Results.** We ran the EBFS and the classical BFS algorithms with the n-queens problem using the representation described earlier with different values of n and summarized the results in the table 1.

Problem	IK	BFS	original EBFS	improved EBFS
5-queens	$(4, 1, -, 5, 2)$	453	372	172
6-queens	$(-, 6, 2, -, 1, 4)$	2 632	1 405	385
7-queens	$(4, 1, -, -, 2, -, -)$	16 831	11 409	11 409
8-queens	$(8, 6, 4, 2, -, 5, 3, -)$	118 878	118 878	118 878

TABLE 1. Comparing the algorithms.

The last three columns show the number of states explored during the search until successful termination. In the case of EBFS, we use two IK states, the initial state and the state described in the second column. Each of the numbers shows the column containing a queen in each rows.

In the first example, the additional IK state is very close to a goal. It shows well the idea behind the improvement. The gain of the original EBFS algorithm is relatively small, because it has to find a full path to the additional

IK state, which needs four forward or four backward operator. The improved algorithm needs only two forward and two backward function.

In the case of the 7-queens problem, four forward operators required to access a goal state. Even though the improved algorithm recognise the connection between the initial state and the additional IK state earlier, it has to continue the search until to a goal state appears.

In the last case, the the additional IK state is not a part of any solution, so the algorithms work in the same way.

5. When to Use the EBFS Algorithm?

The goal of EBFS is to decrease the number of generated nodes by introducing the set of initially known states and starting to explore the representation graph from each of these states. This way, we can explore more than one small subgraph having less nodes together than one bigger subgraph explored by BSF (as illustrated in Fig. 1). The theory works well in cases when we are able to select some useful states from the representation graph, potentially those which appear in the path from an initial state to a goal state. This presumes that we have some extra heuristic knowledge regarding the problem. In other words, we can say that the heuristic knowledge is represented by selecting useful states instead of classifying the states by their worth using a heuristic function. In several cases, enumerating the initially known states is an easier way to represent the heuristic knowledge because there is no need to estimate the worth of all the states, particularly if we only have this information about a few of them.

As we mentioned earlier, the EBFS algorithm is similar to running several BFS algorithms starting from different initial states. However, this has the consequence that the database will now store a part of the representation graph as a graph, not as a tree as in standard BFS. With the standard BFS algorithm, the database can only be reused during a repeated search if the initial state remains the same. In case the initial state changes, the BFS algorithm will build a new tree, discovering edges that have no direct connection with the tree that was built during the previous search. It would be difficult to find out what could be reused from the result of the previous search, even if nearly the same nodes are now discovered. The subgraph stored by the EBFS algorithm is independent of the initial and goal states; it can be reused in its entirety in all cases when the initially known states remain the same. Of course, we can mention several situations when a single run of the EBFS algorithm is much less efficient than the BFS algorithm, but if the discovered subgraph can be reused a number of times, we can gain some execution time. The authors

showed in [3] that in case of the route planning problem based on the real-world public transportation of Budapest, we had to generate much less nodes of the representation graph when using EBFS than by using BFS, despite the fact that we were not able to create the set of initially known states so that EBFS would yield a better result after a single run.

It is important to emphasize that EBFS can only find a solution that contains at least one initially known state. Even in this case, it is not guaranteed that the solution found will be optimal. (Of course, heuristic algorithms are usually not able to find an optimal solution without any restrictions regarding the heuristic estimation. For example, A algorithm does not guarantee to find an optimal solution unless it is actually an A* algorithm.) Our investigation is focused on the number of explored states rather than on finding an optimal solution.

To summarize, the EBFS algorithm is useful in cases when it is difficult to generate states and/or hard to store them because of their size, and thus, we are highly motivated to keep the number of states low. The time complexity of the controller is significantly higher than that of the BFS algorithm.

6. Acknowledgments

The publication was supported by the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 project. The project has been supported by the European Union, co-financed by the European Social Fund.

Many thanks to the reviewer for the insightful comments on this paper.

References

[1] Kádek Tamás, Pánovics János: *Extended Breadth-First Search Algorithm*, International Journal of Computer Science Issues (2013) **10** (6), No. 2, pp. 78–82.

[2] Kádek Tamás, Pánovics János: *Általános állapottér modell*, 23rd International Conference on Computers and Education 2013, Alba Iulia, Romania, pp. 294–299.

[3] Kádek Tamás, Pánovics János: *Extended Breadth-First Search in Practice*, 9th International Conference on Applied Informatics, Eger, Hungary, January 29–February 1, 2014.

Faculty of Informatics, University of Debrecen, 26 Kassai Way, H–4028, Debrecen, Hungary

E-mail address: {kadek.tamas,panovics.janos}@inf.unideb.hu