Proceedings of the 9th International Conference on Applied Informatics
Eger, Hungary, January 29–February 1, 2014. Vol. 1. pp. 59–66
doi: 10.14794/ICAI.9.2014.1.59

Extended breadth-first search algorithm in practice*

Tamás Kádek, János Pánovics

University of Debrecen, Faculty of Informatics
kadek.tamas@inf.unideb.hu, panovics.janos@inf.unideb.hu

Abstract

Classical AI search algorithms are not applicable in practice when the state space contains even only a few tens of thousands of states. 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 and to use a heuristic concept which is easier to embed into search algorithms. However, the base idea of EBFS – i.e., running more explorations of parts of the representation graph starting from several distinct nodes – also implies the possibility of parallelization. In our paper, we show some real-life examples of problems that can be used to illustrate the advantages of EBFS over the classical search algorithms and the use of extended state-space model (ESSM), which was introduced as a possible problem representation technique for EBFS.

Keywords: artificial intelligence, state-space representation, extended model, breadth-first search

MSC: 68T20

1. Extended State-Space Model

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. The solution of the problem is considered the necessary steps required to access one of the goal states from the initial state. The state-space representation of a problem can be illustrated by a directed graph, where each state is assigned to a node, and each edge shows the accessibility of a node from another node, defined by the operators.

*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.

The classical state-space representation can be given as a 4-tuple of the form

$$\langle S, s_0, G, O \rangle ,$$

where

- S is a set of states (the state space), such that $S \neq \emptyset$,
- $s_0 \in S$ is the initial state,
- $G \subset S$ is the set of goal states,
- $O = \{o_1, o_2, \ldots, o_n\}$ is the set of operators, where $o_i \in S^S$.

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.

1.1. Abstraction

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 states, such that $K \subseteq S$ and $K \neq \emptyset$,
- initial $\in \{\text{true}, \text{false}\}^S$ is a Boolean function that selects the initial states,
- 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.

Using the model described above, we are able to represent heuristic with the help of initially known states instead of creating approximations for all of the states. It is enough to enumerate some states that seem to be useful for the search. The model also provides us with the possibility to use both forward and backward searches, not to mention that we can give not only more than one goal state but also more than one initial state.

For more details, see [1].

2. The EBFS Algorithm

A usual way for describing search systems is to discuss the database, the production rules, and the controller separately. In this section, we give a definition of our EBFS algorithm using this approach.

2.1. Database

During the search, the algorithm explores a subgraph of the representation graph. In the general case, the database is a working memory that stores a part of this subgraph. Usually, the more complex the database, the smarter the algoritm. While the "trial and error" method stores only one state and can produce nothing else but a goal state, the A algorithm stores a whole subtree during the search, and it is able to find the solution (or it can recognize that the problem is not solvable).

EBFS stores a subgraph which contains all the nodes representing the initially known (IK) states. For all node n, we need to store the connection (distance) between n and each IK state, the child and parent nodes of n, and the status of n in both directions. The distance between a node n and an IK state k may be defined as the number of function applications required to reach n from k (using forward functions) or k from n (using backward functions) by default. This means that for storing the forward and backward distances for a node, we need two vectors of length l, where l is the number of IK states.

2.2. Production rules

The production rules operate on the database. As in the case of BFS, we have only one production rule, called expansion. A node that is open in a certain direction can be expanded in that direction. For expansion, a node and a direction has to be chosen, and the node has to be open in that direction. Each node has a minimum distance for both directions. The minimum distance of the selected node and direction must be the lowest, considering only the distances of nodes in their open directions.

Whenever a node that is already present in the database is reached during expansion, an update process may be required. The update process will update the distance values in the database pertaining to nodes in the subtree starting from the current node. Note that because all the children and parents are stored, this update can be easily done.

2.3. Controller

The controller describes how the production rules are used to maintain he database. The algorithm can be briefly presented by the following steps:

1. The controller initializes the database: the IK states appear as open nodes in both directions, all distance values are infinite except the ones that belong to the node itself, in which case they are 0.

2. If there are no open nodes in any directions, then the algorithm fails, otherwise continue at step 3.

3. Select a node and a direction for expansion, and expand it.

4. If there is an IK state that can be reached from an initial state and from which a goal state is accessible, then the algorithm terminates successfully with a solution, otherwise continue at step 2.

This is a simple iteration until the algorithm stops with or without a solution. Note that EBFS can only find a solution if it contains at least one IK state. Also note that BFS is a special case of EBFS, where there are only forward functions, and there is only one initially known state, the single initial state.

3. Samples

In this section, we illustrate the benefits of the ESSM model and the EBFS algorithm using three examples of different kinds.

3.1. Three Jars Problem

Problem 3.1. *We have 5 units of water and three jars of different sizes. The first jar can hold exactly 5 units of water and is initially full. The other two can hold 3 and 2 units, respectively, and are initially empty. The task is to measure exactly 4 units of water in the first jar, with respect to the following rules:*

- *We cannot pour water out of tha jars, nor can we add water to the system, we can only pour water from one jar into the other.*
- *We can only pour as much water as possible from one jar into the other. This amount is at most the amount found in the source jar and at most the free space in the destination jar.*

This sample is used only to illustrate the use of the ESSM model, as EBFS was developed to work on state spaces with a huge number of states.

3.1.1. State Space

In this state-space representation, a state of the problem is represented by a 3-tuple, the elements of which describe the actual amounts of water in each jar.

$$A_1 = \{0, 1, \ldots, 5\} \quad A_2 = \{0, 1, 2, 3\} \quad A_3 = \{0, 1, 2\}$$

$$S = \{\langle a_1, a_2, a_3\rangle \in A_1 \times A_2 \times A_3 \ : \ a_1 + a_2 + a_3 = 5\}$$
$$K = \{\langle 5, 0, 0\rangle, \langle 2, 1, 2\rangle\}$$

$$\text{initial}(\langle a_1, a_2, a_3\rangle) = \begin{cases} true & \text{if } \langle a_1, a_2, a_3\rangle = \langle 5, 0, 0\rangle \\ false & \text{otherwise} \end{cases}$$

$$\text{goal}(\langle a_1, a_2, a_3\rangle) = \begin{cases} true & \text{if } a_1 = 4 \\ false & \text{otherwise} \end{cases}$$

3.1.2. Operators Over the State Space

$$F = \{f_{i,j} \in (2^S)^S \ : \ i \in \{1,2,3\} \wedge j \in \{1,2,3\} \wedge i \neq j\}$$

$$f_{i,j}(\langle a_1, a_2, a_3\rangle) = \begin{cases} \emptyset & \text{if } a_i = 0 \vee a_j = \max(A_j) \\ \langle a'_1, a'_2, a'_3\rangle & \text{otherwise} \end{cases}$$

$$\text{where } \ a'_l = \begin{cases} a_l + k & \text{if } l = j \\ a_l - k & \text{if } l = i \\ a_l & \text{otherwise} \end{cases}$$

$$\text{and } \ k = \min(\{a_i, \max(A_j) - a_j\})$$

$$B = \emptyset$$

$$h(\langle a_1, a_2, a_3\rangle) = \begin{cases} 0 & \text{if goal}(\langle a_1, a_2, a_3\rangle) \\ 4 - \sum_{i=1}^{3} \text{sgn}(a_i) & \text{otherwise} \end{cases}$$

3.1.3. Results

- The heuristic function can be replaced with the initially known state $\langle 2, 1, 2\rangle$, which seems to be useful.
- From a total of 12 states, the BFS algorithm explores 8 states, whereas EBFS explores only 7.

3.2. n-Queens Problem

Problem 3.2. *There is an empty chessboard of size $n \times n$. The task is to place n queens on the chessboard so that no queens are attacked by any other.*

There are several good representations for this well-known problem, but now we intentionally use one with a huge number of states in order to illustrate the benefits of the EBFS algorithm.

3.2.1. State Space

In this state-space representation, a state of the problem is represented by an $n \times n$ matrix, in which each cell describes the number of queens in that cell. For

demonstrational purposes, we allow to place the queens on the chessboard in any order, thus resulting in a huge state space.

$$S = \left\{ a = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \in \{0,1\}^{n \times n} \ : \ \mathrm{cond}(a) \right\}$$

$$\begin{aligned} \text{where } \mathrm{cond}(a) = \ & \forall i \, \forall j \, \forall k \, \forall l \, ((i \neq k \vee j \neq l) \wedge a_{i,j} = 1 \wedge a_{k,l} = 1 \supset \\ & \supset i \neq k \wedge j \neq l \wedge |i-k| \neq |j-l|) \end{aligned}$$

$$K = \left\{ \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \right\}$$

$$\mathrm{initial}\left(\left[a_{i,j}\right]\right) = \forall i \, \forall j \, (a_{i,j} = 0)$$

$$\mathrm{goal}\left(\left[a_{i,j}\right]\right) = \forall i \, \exists j \, (a_{i,j} = 1)$$

3.2.2. Operators Over the State Space

$$F = \{ f_{p,q} \in (2^S)^S \ : \ p \in \{1, 2, \ldots, n\} \wedge q \in \{1, 2, \ldots, n\} \}$$

$$f_{p,q}\left(\left[a_{i,j}\right]\right) = \begin{cases} \emptyset & \text{if } \exists k \, \exists l \, (a_{k,l} = 1 \wedge \\ & \quad \wedge (k = p \vee l = q \vee |k-p| = |l-q|)) \\ \left(\left[a'_{i,j}\right]\right) & \text{otherwise} \end{cases}$$

$$\text{where } \ a'_{i,j} = \begin{cases} 1 & \text{if } i = p \wedge j = q \\ a_{i,j} & \text{otherwise} \end{cases}$$

$$B = \emptyset$$

3.2.3. Results

- The heuristic function is replaced with some initially known states, including the initial state.
- Table 1 shows the number of states explored by EBFS using different values of n and different numbers of inititally known states. BFS is actually the same as the EBFS algorithm with only one initially known state, the initial state. The difference between the last two columns is that the three initially known states contain one that is useless during the search for the solution. We can say that this third IK state represents a false heuristic value, but it has little influence on the number of explored nodes.

Problem	BFS	EBFS with 2 known states	EBFS with 3 known states
5-queens	453	216	220
6-queens	2,632	1,409	1,417
7-queens	16,831	4,434	4,439
8-queens	118,878	46,286	46,319

Table 1: Number of states explored by EBFS

3.3. Route Planning Problem

Problem 3.3. *In this problem, we have to find a route from a source stop to a destination stop in the real-world public transportation network of the city of Budapest.*

3.3.1. State Space

Here we use the natural graph representation based on the downloaded GTFS (General Transit Feed Specification) data.

$$\begin{aligned}
\mathcal{G} &= \{\mathcal{N}, \mathcal{E}\}, \quad \text{where } \mathcal{N} \neq \emptyset \text{ and } \mathcal{E} \subset \mathcal{N} \times \mathcal{N} \\
\mathit{source} &\in \mathcal{N} \\
\mathit{destination} &\in \mathcal{N} \\
S &= \{n \ : \ n \in \mathcal{N}\} \\
K &= \{n \ : \ n \in S \wedge \operatorname{card}(\{(n_1, n_2) \in \mathcal{E} \ : \ n_1 = n \vee n_2 = n\}) \geq \epsilon\} \\
\operatorname{initial}(a) &= (a = \mathit{source}) \\
\operatorname{goal}(a) &= (a = \mathit{destination})
\end{aligned}$$

3.3.2. Operators Over the State Space

$$\begin{aligned}
F &= \{f\}, \text{ where } \ f(a) = \{a' \in S \ : \ (a, a') \in \mathcal{E}\} \\
B &= \{b\}, \text{ where } \ b(a) = \{a' \in S \ : \ (a', a) \in \mathcal{E}\}
\end{aligned}$$

3.3.3. Results

- The heuristic function is now replaced with some initially known states representing some stops with the highest number of connections.
- Despite of our expectations, the BFS algorithm kept finding a solution with less explored states than EBFS.

- In this problem, we could make use of the reusability of the database. Table 2 summarizes the results of this:

Number of searches	Total gain	Number of misses	Number of searches	Total gain	Number of misses
1	-104	0	10	7,392	0
2	-1,344	0	100	100,934	0
3	-35	0	1,000	969,241	2
4	382	0	10,000	9,707,504	8

Table 2: Total gain in the number of explored states after a series of searches

BFS is actually the same as the EBFS algorithm with only one initially known state, the initial state. The difference between the last two columns is that the three initially known states contain one that is useless during the search for the solution. We can say that this third IK state represents a false heuristic value, but it has little influence on the number of explored nodes.

References

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

[2] Beamer, S., Asanović, K., Patterson, D.: Direction-Optimizing Breadth-First Search, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (SC '12), Salt Lake City, Utah, November 2012, Article No. 12.

[3] Pineau, J.: Tractable Planning Under Uncertainty: Exploiting Structure, doctoral (Ph.D.) dissertation, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2004.

[4] Edelkamp, S., Schrödl, S.: Heuristic Search — Theory and Applications, Morgan Kaufmann Publishers, Waltham, Massachusetts, 2012.