TRANSNAV
http://www.transnav.eu
the International Journal
on Marine Navigation
and Safety of Sea Transportation
Volume 8
Number 4
December 2014

DOI: 10.12716/1001.08.04.11

Learning Search Algorithms: An Educational View

A. Janota, V. Šimák & J. Hrbček
Faculty of Electrical Engineering, University of Žilina, Slovakia

ABSTRACT: Artificial intelligence methods find their practical usage in many applications including maritime industry. The paper concentrates on the methods of uninformed and informed search, potentially usable in solving of complex problems based on the state space representation. The problem of introducing the search algorithms to newcomers has its technical and psychological dimensions. The authors show how it is possible to cope with both of them through design and use of specialized authoring systems. A typical example of searching a path through the maze is used to demonstrate how to test, observe and compare properties of various search strategies. Performance of search methods is evaluated based on the common criteria.

1 INTRODUCTION

The ability to create and use an internal machine model of the world is an important feature of "intelligent" systems profiting from artificial intelligence (AI) components. If initial and goal models of the world are available, the AI system is expected to solve the problem, i.e. to find a proper sequence of actions determining how to get from the initial to the goal model. Since in many cases a certain level of abstraction is necessary to simplify the problem, one can take advantage of state space representation. Thus the state space can be seen as a set of states of the problem to which we can get by applying operators to a particular state of the problem to get a new state (or remain in the same state). Transitions between models may be represented by transitions between states. To represent a state space one may use an oriented graph consisting of vertices (nodes) and edges. Each vertex of the graph is a complete description of the state. The state space can be huge due to the combinatorial explosion. Problem solution then may be formulated as a search of the path through a directed graph under fulfilling certain conditions. Those states in which more transitions (rules) might potentially be applied bring conflicts. The way of how the conflict could be solved is defined by the used control mechanism (problem solving method). Tasks solved in the real world may cover searching paths, optimizing electrical circuits, cryptography, planning, configuring routes, etc. Generally, there are different kinds of tasks (problems) to be generally addressed: mundane tasks such as perception (e.g. vision, speech), natural language processing (e.g. understanding, generation, translation), common sense reasoning or robot control; formal tasks such as games (e.g. chess, checkers, Sudoku, etc.) or mathematics (e.g. geometry, logic, integral calculus etc.); and expert tasks such as engineering (e.g. design, fault finding, manufacturing planning, etc.), scientific analysis, medical analysis, financial analysis, and many others [1]. Considering relevance of the presented topic to the main area TransNav interests, we can mention a general survey of AI applications to critical transportation issues [2] or particular problems of maritime surveillance

modelling (e. g. testing of backtracking algorithms [3] or, treating the problem as a variation of a Travelling Salesman Problem [4]).

The AI courses at the universities are provided based on standard sources such as e.g. [5], however many different approaches to teaching AI can be found around the world. Several introductory programming courses use problems in AI as motivating examples [6][7]; the AI is often being taught through the games to attract more students into computing [8][9]. Motivation is a key factor in the learning process [10]. To make students properly motivated, specialized education software tools can be developed and used to demonstrate selected search strategies through typical problems such as searching path through the maze, shortest path in a graph, Sudoku, etc. In this paper an authoring tool, called "Labyrinth", is used as a working example. The Labyrinth application has been implemented in both declarative and procedural ways (SWI Prolog and Java) and helps to demonstrate problem solving and to obtain statistic data needed for further evaluation. The set of tested methods includes the most usual search algorithms which may be tested, observed and their properties compared not only theoretically, but also practically. Performance of particular searching methods in a state space is evaluated based on the common criteria, i.e. completeness, time complexity, space complexity and optimality.

2 DESIGN OF THE AUTHORING SOFTWARE SYSTEM

2.1 *General Formulation of the Task*

Persons who want to know how to solve tasks from above categories must master perceptual, linguistic and common sense skills, followed by skills characterizing the application domain. On the ontological level a problem-solver consists of the five major elements [11]:

1 A problem-solving goal.
2 Domain data describing the problem instance.
3 Problem-solving state.
4 Problem solving knowledge.
5 Domain factual knowledge.

The emphasis is given to search as the primary technique used in Computer Science and Operations Research for solving computation-intensive combinatorial optimization problems, typically those in the NP-hard class [12]. State space searching has a number of interesting properties, such as the ability to guarantee optimal solutions and the possibility of exploiting domain knowledge to guide the search [13]. The core of the search problems is "How to control the search?". At the beginning there is given:

1 An Initial State that the problem starts in.
2 A set of operators that can be taken.
3 A Goal State (or a set of Goal States) that the problem ends in.
4 Optionally, paths Cost function to solve the problem.

The task is formulated as finding a sequence of operators leading from the Initial State to the Goal State. The search space is a tree (graph) defined by the initial state and the operators. The search tree (graph) is an explicit tree generated during the search by the control strategy. Different search algorithms use different search strategies. Generally, they may be either uninformed (make no use of domain knowledge - also known as blind search) or informed (using some rule(s) of thumb). The task could be expressed as a General Search algorithm, consisting of the following steps:

1 Initialize the search tree with the initial state.
2 Report failure if search tree is empty.
3 Move to a leaf node according to a strategy.
4 Ready if a goal state.
5 Expand the current state by generating successors to the current state. Add them to the search tree as leaves.
6 Repeat from 2.

For the sake of simplicity the more formal (mathematical) definitions of algorithms and state spaces have been intentionally avoided here since they are available in many AI textbooks. Generally, the search strategies apply the following four criteria:

1 Completeness: is finding a solution guaranteed?
2 Time complexity: How long do they take in search time? (What is a number of generated nodes?).
3 Space complexity: Storage required? (How many nodes are to be stored in a memory?).
4 Optimality: When there are several solutions, does it find the best one?

2.2 *The Problem*

Representing search algorithms through the state of a vertex is difficult since it is constantly changing (e.g. a vertex can be unexplored, open or closed) and the audience easily loses the time-line sequence. Thus the problem has two levels, one that is technical and another that is psychological [14].

There are different approaches how to cope with the technical side of the problem – to use colourful figures; to visualize a changing tree through multimedia means (animations, videos); or to apply interactive programs solving particular problems. Very often animated figures are used as a part of presentations, where the algorithms and the graphs can be seen developed step-by-step, differentiating state changes by different colours, symbols, etc. What is more, various specialized software tools are available to visualize explained search algorithms..

The psychological level of the problem may result from requirements to use programming knowledge when creating a state-space representation along with the attached data structures and selecting the appropriate algorithm that is recursive in many cases. Since the length of the solution is not determinable beforehand, its storing requires a dynamic data structure [14].

Potential solution of both sides of the learning problem can be seen in the two-steps approach: the first step consists in learning a theory together with practical application of selected search algorithms to a certain problem, using pre-designed software tools for both managed and self-study evaluation of basic properties. The seconds step covers step-by-step solution (programming) of problems with gradually

growing complexity by already known search algorithms. For that purpose a basic course in a proper programming language (in our case SWI Prolog) must be provided. Finally, obtained knowledge may be utilised in own individual projects.

Obviously, modifications and depth of provided knowledge depends on what kind of future profession is to be addressed (electrical engineering, computer engineering, transport engineering, etc.). Details on trajectories of electrical engineering and computer engineering students by race and gender can be found in [15].

2.3 *System Concept*

The attention here is paid to the application called Labyrinth. The problem consists in finding a path through the maze. Considering the facts mentioned above, the application should make us possible to generate a labyrinth structure of the required size (different dimension/complexity of the problem). Then, for the generated configuration, the Initial State must be defined in such a way that all crossing and terminal points are given letters representing nodes in the state space (Fig. 1).

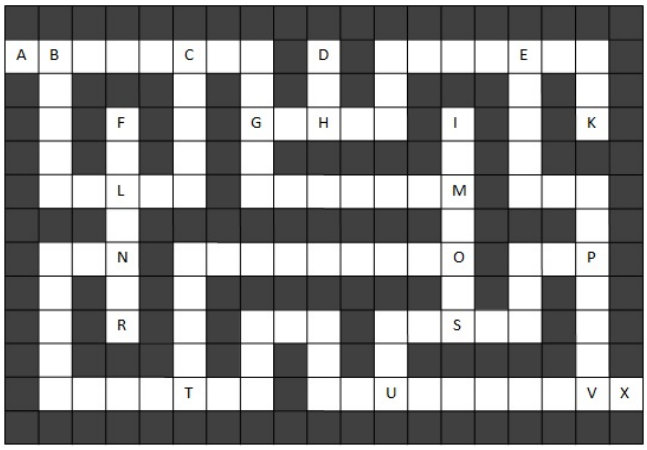

Figure 1. Example of marking nodes in a generated Labyrinth configuration

This makes possible to create a model of the state space. The nodes (vertices) represent names of the crossings and terminal points of paths and transition values correspond to distances between the nodes (Fig. 2).

Two versions of the authoring tool Labyrinth have been developed so far: the simpler version which provides only 3 types of algorithms - Breadth-First Search (BFS), Depth-First Search (DFS) and Bi-directional Search (BS), and the more complex version that also involves Iterative-Deepening (ID), Backtracking (BT), Uniform Cost Search (UCS), Greedy Search (GS), A*, and Dijkstra algorithms.

As the first step it is necessary to make initial settings of the Labyrinth application. The more complex version includes:

1 Selection of the search algorithm types to be used.
2 Definition of the labyrinth dimensions (width * height).
3 Definition of the step size (for statistics calculations).
4 Enabling or disabling the option "Find more solutions".
5 Accuracy of obtained data.

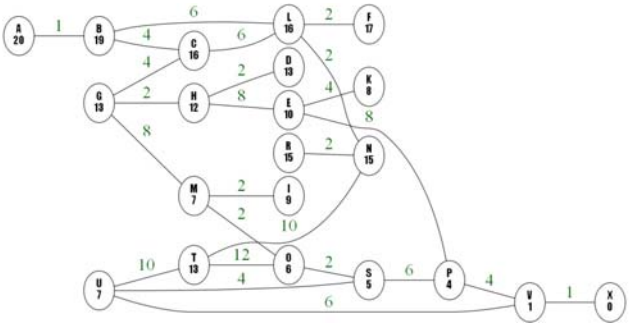

Figure 2. Example of marking nodes in a generated Labyrinth configuration

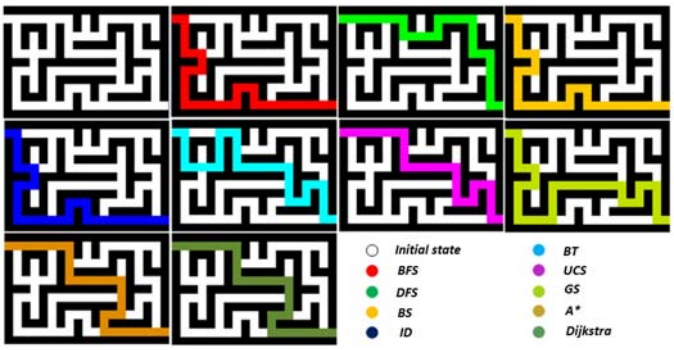

Figure 3. Different solutions found by the Labyrinth application

Based on the settings above it is possible to generate the maze structure and write relevant data into the log file. All solutions found for particularly used search strategies can be visualized and seen in their graphical representations (Fig. 3).

2.4 *Implementation*

The Labyrinth tool has been developed in several versions (simpler and more complex), and also using different programming approaches - procedural and declarative ones.

The Java version (Fig. 3, 4) plays the motivation role and makes possible to show and evaluate basic parameters of mentioned search strategies applied to the Labyrinth problem in the basic part of the course. The application was created in the Eclipse environment (Eclipse Standard/SDK, Version: Kepler Service Release 1). To run it the Java package (min. version 7, update 45) must be installed.

As indicated above, at the beginning the user may define the required size of the Labyrinth through the "height * width" setting available from the main window (Fig. 4a) and after confirming his/her settings automatically generate its random structure. Then there is a chance to set other preferences that determine the range of evaluated statistic data (Fig. 4b): single or multiple solutions, required types of the search algorithms, accuracy, min. and max. size of the Labyrinth and the step size. Then it is possible to run

the search and get visualized solution(s) and all mentioned statistics in the form of tables (Fig. 4c) where the parameters are observed for different Labyrinth sizes (here from 11x11 to 19x19 with the step 2). At the moment the application uses the Slovak menu only (therefore some figures in this text were modified and/or supplemented with English descriptions).

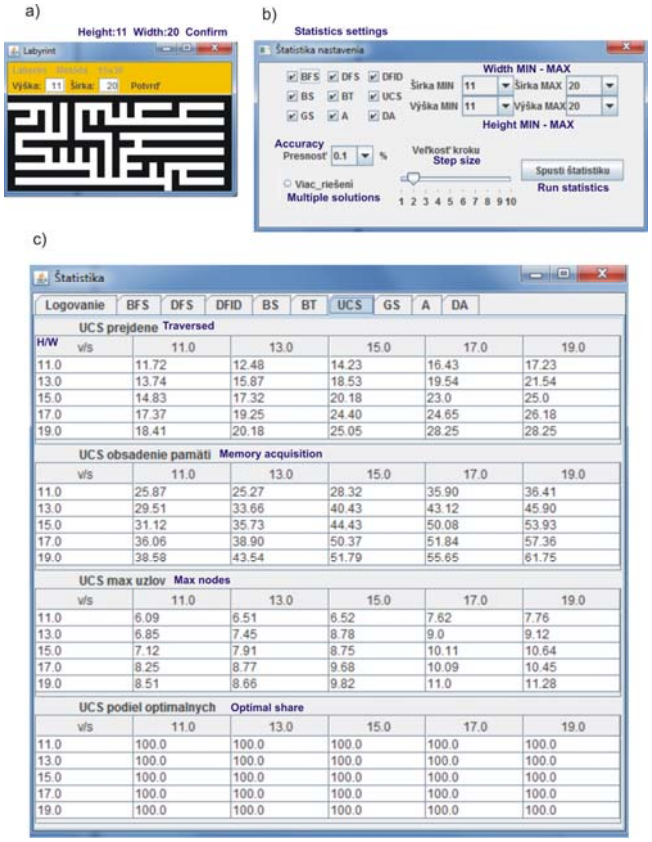

Figure 4. The samples of the Labyrinth user interface display (for the size 20x30)

The second version of the Labyrinth application has been designed using the SWI-Prolog 6.0.0. Since it assumes knowledge of the predicate calculus logics and principles of declarative programming, the students must be first provided with the background knowledge and the practical programming skills. That is ensured in the following advanced courses. Then it is possible to build the application gradually, together with the students and for particular search algorithms. A part of the source code for the simpler version performing the BFS, DFS, and BS is shown in Fig. 5.

2.5 *Evaluation and Analysis*

As seen above, for all selected algorithms the menu items are created together with tables showing a number of traversed (processed) nodes for the defined sizes of the labyrinth, the total number of nodes stored in the memory, the maximum number of nodes to be processed at the moment and the share of solutions with the minimum path. Data obtained in this way may be used to show dependence of traversed nodes on the labyrinth dimensions, share of the shortest possible solutions and usage of memory by the given algorithm.

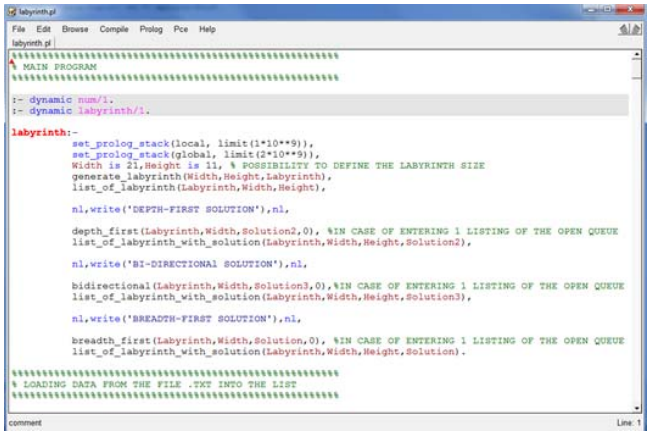

Figure 5. A part of the SWI-Prolog source code for the Labyrinth application (with BFS, DFS, BS)

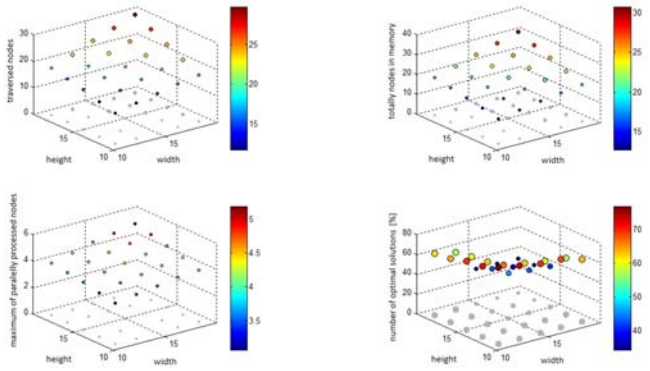

Figure 6. Graphical representation of the search algorithm characteristics (BFS example)

To explain the meaning of data obtained in the table form (Fig. 4c), the characteristics of the individual search algorithms could be presented in the graphical form (Fig. 6).

For the sake of mutual comparison of time complexity statistic data might be processed and presented in the form of a graph, elaborated e.g. in the Matlab environment (Fig. 7). Thus one could deduce that the most time demanding methods seem to be ID and BT algorithms. For the labyrinth size 100 x 100 they traverse more than 30 000 (ID), resp. 4 000 nodes (BT). Time complexity grows exponentially. They are followed by the UCS algorithm which has a little bit less steep increase of search time and for maximal sizes of the labyrinth it traverses approximately 2500 nodes. Time complexity of other tested methods is almost the same, i.e. in the order of hundreds nodes for the maximal labyrinth size (about 200 traversed nodes for the GS up to 800 nodes for the Dijkstra algorithm).

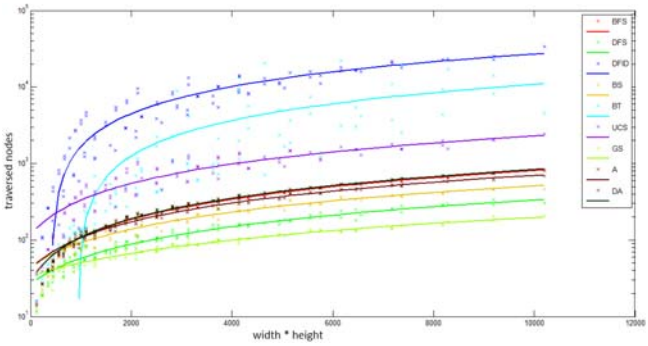

Figure 7. The number of traversed nodes depending on the Labyrinth size

To evaluate space complexity one must take into account difference between the totally consumed memory space (Fig. 8) and the actually allocated space in the operation memory (Fig. 9).

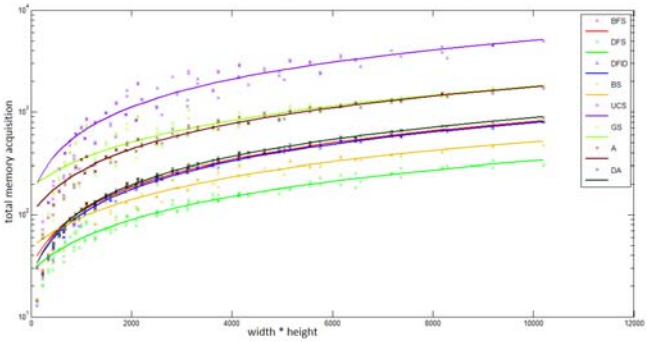

Figure 8. Total operational memory acquisition depending on the Labyrinth size

For considerably big state spaces (unlike our Labyrinth application) data could be stored to a permanent memory (e.g. hardisk) and thus to save operation memory. While methods of the best first search (UCS, GS, A*) show considerable total memory consumption (if compared with uninformed methods) their consumption of operation memory is comparable. The most space demanding method is the Dijkstra algorithm which works with all nodes in the state space during every cycle (for the backtracking algorithm the space complexity has not been evaluated since in the Labyrinth application it was implemented with the min. space complexity, i.e. working with 1 node in the memory only). Since the uninformed methods cannot find the shortest path through the maze from the viewpoint of its length but only from the viewpoint of number of nodes, the only acceptable methods in the application remain the UCS, A*, and Dijkstra algorithms.

The given graph (Fig. 10) indicates that uninformed methods with growing size of the state space very quickly loose capability of finding an optimal path. On the other side, if only 1 solution exists, this property becomes irrelevant (i.e. if only 1 path is available in our Labyrinth application). Globally, the small space complexity of uninformed methods could be confirmed; however, their usability is limited to those tasks where optimal solution is needed or tasks in which they are able to provide acceptable solution without any additional information. Within the all acceptable methods the A* algorithm seems to be the best if considering its time complexity because its space complexity is also acceptable.

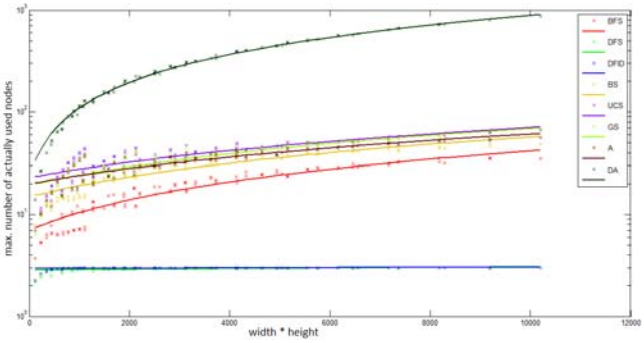

Figure 9. Actual operational memory acquisition depending on the Labyrinth size

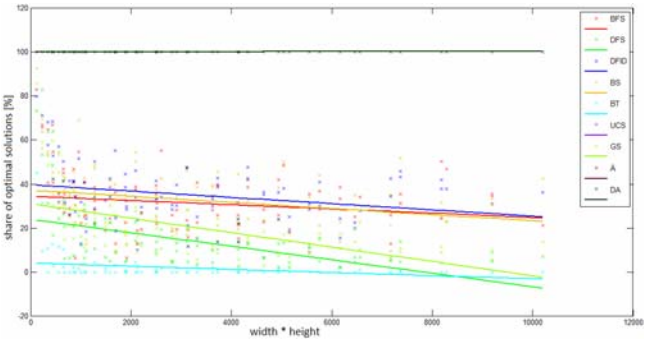

Figure 10. Share of the optimal solutions (optimality criterion: the path length)

3 ACKNOWLEDGMENTS

This work was supported by the Slovak grant agency KEGA under the project “KEGA 010U-4/2013 Modernization of didactic equipment and teaching methods with a focus on the area of robotics". The authors thank M.S. Ivan Sakál for his technical help and software support.

4 CONCLUSION

The paper demonstrated an educational aspect of how formal tasks solved through search algorithms could be modelled, tested and evaluated. As a working example the Labyrinth authoring software tool has been discussed, implemented in both declarative and procedural ways (SWI Prolog and Java). It helped to obtain statistic data usable for evaluation of applied methods. The set of tested methods included most usual search algorithms.

REFERENCES

[1] R. A. Akerkar, and P. S. Sajja. Knowledge-Based Systems. Jones & Bartlett Learning, 2010.
[2] Artificial Intelligence Applications to Critical Transportation Issues. TRC E-C168. November 2012. http://onlinepubs.trb.org/onlinepubs/circulars/ec168.pdf
[3] D. O. Marlow and J. E. Murphy. Testing various backtracking algorithms in airborne maritime

surveillance modelling. In 20th International Congress on Modelling and Simulation, Adelaide, Australia, 1–6 December 2013. http://www.mssanz.org.au/modsim2013/D1/marlow.pdf
[4] Kilby, P., Tobin, P., Luscombe, R., Barry, S. and Hickson, R. The maritime surveillance problem. In T.R. Marchant, M. Edwards and G.N. Mercer (eds.), Proceedings of the 2007 Mathematics-in-Industry Study Group, 32-56, 2008
[5] S. Russell and P. Norvig. Solving problems by searching in Artificial Intelligence. A Modern Approach, 3rd ed. New Jersey: Prentice Hall, 2010, ch. 3, pp. 64–119.
[6] D. S. Touretzky. Preparing computer science students for the robotics revolution. Communications of the ACM. 53(8): 27-29, 2010.
[7] T. W. Neller, C. G. Presser, I. Russell, and Z. Markov. Pedagogical possibilities for the dice game pig. J. Comput. Small Coll. 21(6), 149-161, 2006.
[8] D. Wong, R. Zink, and S. Koenig. Teaching Artificial Intelligence and Robotics via Games (Abstract). In AAAI Symposium on EAAI 2010, http://www.cs.huji.ac.il/~jeff/aaai10/02/AAAI10-342.pdf
[9] M. Zyda and S. Koenig. Teaching artificial intelligence playfully. In Proc. AAAI-08 Education Colloquium, pages 90-95, 2008.
[10] P. J. Muñoz-Merino, M. F. Molina, M. Muñoz-Organero and C. D. Kloos. Motivation and Emotions in Competition Systems for Education: An Empirical Study. IEEE Transactions on education. 57(3): 182-187, 2014.
[11] B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins. What are ontologies, and why do we need them? IEEE Intelligent Systems. 1(4): 20–26, 1999.
[12] W. Zhang. State Space Search for Problem Solving. State Space Search. Algorithms, Complexity, Extensions, and Applications. 1st ed. New York: Springer-Verlag, 1999.
[13] R. Varela and E. Soto. Scheduling as heuristic search with state space revolution. Lecture Notes in Computer Science. 2527: 815-824, 2002.
[14] G. Kovásznai and G. Kusper. Artificial Intelligence and its teaching. 1st ed., 1990 http://aries.ektf.hu/~gkusper/ArtificialIntelligence_LectureNotes.v.1.0.4.pdf
[15] S. M. Lord, R. A. Layton, and M. W. Ohland. Trajectories of Electrical Engineering and Computer Engineering Students by Race and Gender. IEEE Transactions on education. 54(4): 610-618, 2011.