International Journal of Internet, Broadcasting and Communication Vol.11 No.3 49-58 (2019)
http://dx.doi.org/10.7236/IJIBC.2019.11.3.49

IJIBC 19-3-7

Design of Problem Solving Primitives for Efficient Evidential Reasoning

Gye Sung Lee

Dept. of Software, Dankook University, Korea
gslee@dankook.ac.kr

Abstract

Efficient evidential reasoning is an important issue in the development of advanced knowledge based systems. Efficiency is closely related to the design of problems solving methods adopted in the system. The explicit modeling of problem-solving structures is suggested for efficient and effective reasoning. It is pointed out that the problem-solving method framework is often too coarse-grained and too abstract to specify the detailed design and implementation of a reasoning system. Therefore, as a key step in developing a new reasoning scheme based on properties of the problem, the problem-solving method framework is expanded by introducing finer grained problem-solving primitives and defining an overall control structure in terms of these primitives. Once the individual components of the control structure are defined in terms of problem solving primitives, the overall control algorithm for the reasoning system can be represented in terms of a finite state diagram.

Keywords: *Knowledge Based System, Efficiency, Evidential Reasoning*

1. Introduction

Knowledge-based system technologies in the past years have reached a level of sophistication that enables their applications to complex real-world scenarios. In such systems, performance is the key to its effectiveness. Performance can be viewed from different perspectives, but two important aspects of performance that we focus on in this research are efficiency and reliability. Reliability refers to the system's capability for drawing the right conclusions in different problem-solving situations. As a step towards achieving reliability in the framework of efficiency, a number of machine learning algorithms are adopted to build rule models. Efficiency pertains to the computational aspects of the reasoning process, i.e., how quickly can the system generate an appropriate conclusion to a given problem. A number of methodologies have been developed for improving problem-solving efficiency [1-3]. In this research, we adopt two major schemes: (i) reasoning mechanisms based on problem-solving methods (PSM's) and (ii) reorganization of the knowledge structures of an existing knowledge base to improve problem-solving performance.

The explicit modeling of problem-solving structures is the key to efficient and effective reasoning. It has

Manuscript Received: June. 3, 2019 / Revised: June. 9, 2019 / Accepted: June. 14, 2019
Corresponding Author: gslee@dankook.ac.kr
Tel: +82-10-3344-7529, Fax: +82-31-8005-3223
Department of Software, Dankook University, Korea

been also pointed out that the problem-solving method framework is often too coarse-grained and too abstract to specify the detailed design and implementation of a reasoning system. Therefore, as a first step in developing a new reasoning scheme based on a problem-solving architecture, the problem-solving method framework is expanded by introducing finer grained problem-solving primitives and defining an overall control structure in terms of these primitives.

In our reasoning structure, this problem is handled with the provision of more detailed problem-solving primitives for detailed subtasks and these are configured to explicitly express the overall control structure. This clear and modular definition of subtasks for the given problem and explicit control structures benefit system designers and knowledge engineers in designing and implementing a knowledge-based system. That is, this design framework can allow them to speed up the system design process and make effective in maintaining the knowledge-based system. More importantly, to improve the efficiency of the reasoning system, our reasoning scheme utilizes the domain expert provided knowledge base by reorganizing it to best fit the particular problem-solving task. The restructured problem-solving knowledge, called rule model, is structured as a hierarchy of concepts in the knowledge base. The rule model is created by use of machine leaning algorithm over rules and used to build the context during the reasoning process.

2. Efficient Problem Solving

2.1 Problem Solving Methods

A key issue in building efficient systems is how to deal with the characterization and understanding of the features of the application. This information is then used to acquire relevant problem-solving knowledge and select appropriate problem-solving tools. A number of conceptual frameworks have been developed to model this approach to system building: (i) task theory [1,4], (ii) heuristic classification systems [5], and (iii) problem-solving methods [1] All three concepts focus on the importance of the correspondence between the type of the knowledge and problem-solving methods (PSMs) used in different problem-solving situations. In general, different terminologies are used: problem-solving method, control knowledge, inference procedure knowledge, and strategy knowledge.

Early knowledge-based systems focused on general-purpose inferencing mechanisms, such as data-directed reasoning (forward chaining) and goal-directed reasoning (backward chaining) that were applied to specific domain knowledge to solve problems. In this framework, problem-solving efficiency and effectiveness was achieved by tailoring knowledge base structures in arbitrary ways. In general, this ad hoc structuring of rules and clauses within rules was hard to implement. But more importantly, transparency and the ability to explain problem-solving behavior was lost in the process. Since early knowledge based systems required ad hoc structuring of rules and clauses within rules, a lot of the problem-solving knowledge had to be implicitly encoded into the domain knowledge base itself. Limitations of this approach led to the evolution of more sophisticated and complex control structures, such as the explicit recording of abstract control knowledge and meta-knowledge [6].

Problem solving methods are better exploited for specific tasks (e.g., design, diagnosis) to introduce problem-solving methods (PSMs) both for acquisition of relevant knowledge and for inferencing. For example, for diagnosis task, a PSM called `cover and differentiate' is designed to fit for diagnostic problem-solving. The use of this explicit PSM in this task implies that the system seeks domain knowledge in particular forms to achieve problem-solving goals. The `cover and differentiate' PSM implies that the system requires two kinds of problem-solving knowledge to link hypotheses and symptoms: (i) for given sets of symptoms what set of hypotheses cover or explain these symptoms, and (ii) for given a set of hypotheses what additional information

(observations and tests) could be used to differentiate among hypotheses. Not only does this lead to more directed and efficient reasoning but it is also used in the knowledge elicitation process, i.e., given a problem-solving situation, knowledge is sought that plays a particular role. This idea of ``knowledge roles'' is used to develop a number of knowledge-based systems.

In general, system building involves identifying the problem-solving methods in advance and its roles in acquiring appropriate knowledge. This is an important step towards identifying and simulating the problem-solving reasoning in light of knowledge acquisition and refinement. Another type of task, heuristic classification, can be described in terms of a three step reasoning process:

1. data abstraction - abstract specific symptoms to general categories,
2. heuristic match - match the abstracted data with an appropriate solution class, and
3. solution refinement - refine this solution class to a specific solution that explains the actual data.

This procedure can be summarized as a three step problem-solving method: abstract, match, and refine.

2.2 Task Theory

Similar to the spirit of problem-solving methods which establish a relationship between features of a problem and its solution method, tasks are introduced for an agent to describe desirable environment states that should be brought about [4]. The system is designed to perform tasks that currently require human intelligence. The concept of tasks is closely related to task features with inference structures. The underlying assumption is that knowledge and inference structure can be classified into generic types based on the problem-solving task (e.g., classification, design, diagnosis) that is being addressed.

Tasks have been viewed in terms of three components: problems, methods, and subproblems. A problem normally requires one or more methods to solve it. A method is characterized by a particular form of knowledge, a corresponding inference structure, and a definition of additional subproblems required to solve the problem. Therefore, tasks are developed from a goal-oriented problem-solving viewpoint, and can be described as a combination of a problem, method, knowledge, and inference structure. Examples of typical tasks we encounter in problem-solving are diagnosis, classification, interpretation, and construction. Task characterization is based on the following information: (i) input-output behavior called task function, (ii) domain knowledge used to perform the task, (iii) inferences appropriate for the task, and (iv) method for controlling the knowledge and inferences to solve a task or a subtask. For example, hierarchical classification is a major task in diagnosis. The corresponding problem-solving task involves refining the hypothesis: from most general to specific. Correspondingly, knowledge structures are defined that place hypotheses in a predefined hierarchy from more general to more specific. When control arrives at a specific hypothesis, computations are performed to match data (evidence) to the current set of hypotheses (hypothesis matching). The task formalism can also be used for knowledge acquisition and explanation generation. For knowledge acquisition, tasks help identify what knowledge is needed for an instantiated method which is associated with a given task. For explanation generation, tasks use the initiated method and its inference structure to explain a solution trace.

2.3 PSMs and Tasks

The advantages of role definition using problem-solving methods are summarized as:

1. the system understands the role or function of the knowledge required by the task,

2. conversely, the functionally represented knowledge can recognize the features of the task,
3. understanding the knowledge role lets the system determine what knowledge is appropriate in a given situation and how to apply the gathered knowledge,
4. understanding the knowledge role helps map the description of a problem onto a problem-solving method, and
5. problem solving methods produce better explanations of a system's problem-solving trace.

A variety of problem-solving methods that have been proposed are listed in Table 1.

Table 1. List of Problem Solving Methods

Cover and Differentiate	Abstract, Match, and Refine
Establish and Refine	Acquire and Present
Propose, Filter and Select	Establish and Select
Propose and Revise	Propose and Apply

3. Machine Learning Approach for Efficient Problem Solving

The performance goal of most AI systems is to achieve problem-solving behavior that matches an expert's style of reasoning and performance in generating solution efficiently. In the PSM framework this is achieved by: (i) choosing a particular problem-solving method to fit the characteristics of a particular task so that reasoning mechanisms can be tailored for efficient problem-solving, and (ii) acquiring knowledge from human experts in a form that fulfills specific roles in the problem-solving process. As discussed earlier, this methodology influences: (a) the design of the inferencing structures and (b) the knowledge acquisition process. However, once the inferencing structures are determined and knowledge is acquired, the organizational knowledge structure is static, i.e., it is unchanged by the problem-solving process. On the other hand, machine learning techniques tend to focus on the use of knowledge and problem-solving structures during problem-solving. Analyzing performance may result in reorganization of knowledge structures and problem-solving failures may direct the acquisition of new knowledge. For example, a representative learning scheme is based on skill refinement. This is motivated by the fact that human problem-solving skills are acquired and improved by problem-solving experience. This is developed by the idea of chunking from a similar cognitive observation: people improve their performance by repetitive practice on one task or a set of similar tasks.

Another framework of learning from past experience is learning from examples, explanation-based learning(EBL)[7]. The difference between them is that the former requires a extensive amount of examples, on the other hand, the latter can learn from single examples or a set of cases. EBL's learning concepts from a single example is made possible by the assumption that the concepts can be derived from a complete and consistent set of domain knowledge. In this section, we review learning techniques from the viewpoint of performance improvement during problem-solving.

3.1 Learning by Chunking

A very rudimentary type of human learning is rote learning. Memorizing computed information and associated situations in solving a specific task can improve performance significantly, especially during initial learning[8]. For example, a game playing program may save the heuristic scores computed at each node of the search tree during a particular problem episode. Later, if the same situation arises the program can look up the

saved scores and this can lead to significant performance improvement by avoiding expensive recomputation. This idea can be extended further to a more general problem-solving framework where problem solutions are represented as a sequence of operators. By saving operator sequences that correspond to particular subproblems or problems and revising those in subsequent problem-solving can produce significant performance gains.

The chunking theory of learning is an abstract cognitive model of human learning, i.e., humans improve performance via the acquisition of knowledge about patterns derived from problem-solving tasks. The formation of these patterns, called chunks arise from goal-based experience, where each goal, describing a desired state of affairs, is represented as a data structure. The goal can be decomposed into a set of simpler goals (subgoals) and those subgoals can be decomposed even further. This results in a goal hierarchy. Subgoals have associated problem-solving methods. These methods could be rigid algorithms (domain-specific problem-solving methods) or they could be drawn from more flexible weak methods, such as means-ends analysis and heuristic search.

When a subgoal is successfully solved, the chunking process combines the set of applicable rules into a chunk. The first step of the process involves collecting the parameters of the situation that exists prior to the goal being solved, and the results created after processing the goal. The chunk, a new rule, sets these parameters as the conditions for the rule and the results as the conclusions for the rule. In order for the new rule to apply to a variety of situations, some of the constants in the conditions and conclusions of the rule need to be generalized by replacing the constants with variables. Problem-solving based on chunking occurs as state space traversal. A problem space consists of a set of states representing possible situations and a set of operators that transform one state to another. Decisions made during problem solving can be represented as problem space search. Long-term memory consists of a set of production rules and short-term memory (also called working memory) contains facts derived by rules in long-term memory. Content of short-term memory at a given moment can be mapped to a state of the problem space and is then analyzed for acquiring chunks. Systems based on chunking learn search-control knowledge (rules) and also improve their performance by detecting more useful sequence of rule firings (chunks) through problem-solving.

3.2 Macro-Operators

The process of forming a chunk from a sequence of rule firings is analogous in many aspects to forming macro-operators from a sequence of operators applied to solving a problem. Similar to chunking, macro-operators derive useful sequences of operators by analyzing problem-solving traces. Robot planning uses macro-operators for creating and generalizing action plans. In order to achieve a given goal, a sequence of operators is searched that transforms the problem state. A successful sequence of operators is lumped into a single operator called a macro operator. The macro-operator has two components: preconditions and postconditions. The preconditions captures the initial description of the problem situation and the postconditions corresponds to goal description that has to be achieved from the initial situation. Macro-operators are then generalized by replacing constants in pre- and post-conditions with variables.

Macro-operators in [9] was developed for solving planning problems. This method utilizes structures of plan, which can be used to obtain useful knowledge of the solution. This can help find solutions to more complex planning problem. Originally, macro-operators in early days were developed for a problem solving methodology. This method could solve the planning problem in an efficient way some problems that could not be solved by traditional weak methods. The basic idea of this method was that macro-operators applied to a certain situation may temporarily interfere with previously satisfied subgoals but at the end all the subgoals were restored and the goal of the macro-operators was also achieved. The macro-operators were stored in a

two-dimensional table called a macro table. Subgoals were achieved by looking up the table and applying a single macro-operator. The basic technique involved was to search the space of macro-operators. Three search mechanisms were used to generate the complete set of macro-operators: (i) brute-force search, (ii) bi-directional search, and (iii) macro composition. When a macro was generated through these search steps, it was inserted into the macro table in its correct slot if it was a shorter macro-operator than the existing one in the slot. Like chunking, macro-operators enhanced performance by avoiding recomputation. However, unlike chunking, these macro-operators were constructed by preprocessing the problem-solving task to generate efficient solutions in different situations. On the other hand, chunking incrementally learn from problem-solving experience.

3.3 Explanation Based Learning

As discussed above, the effectiveness and efficiency of chunking is its ability to capture useful patterns from problem-solving experience and then create generalizations from them. Similarly, the explanation based learning(EBL) paradigm learns complex concepts through problem-solving experience and then generalizes them for better definition. However, unlike empirical learning techniques such as inductive learning where multiple instances of a concept are required for learning the concept, EBL is specifically designed for generalizing from a single example. The foundation of EBL has been justified by the observation that people are often able to learn a general rule or concept after observing a single instance of the concept. In the literature, EBL is known as a method of generalizing from examples. The generalization process is usually described in terms of two steps: (i) explain the example, (ii) analyze and generalize the explanation. Given a training example, explanation is generated as a proof that establishes how relevant features satisfy the goal concept.

Domain theory is used to generate the explanation. The domain theory includes definitions of relations, rules, and facts. This explanation generation process using domain theory performs an important job: eliminating the redundant or irrelevant features of the training example. The second step is to generalize the relevant features of the training example and produce general sufficient conditions under which the explanation holds. The technique in EBL traces back the explanation structure established during the explanation generation and computes an expression that represents a sufficient condition for the desired concept definition. This process is called goal regression [10]. In this process, an important factor is variable bindings. Variables in the goal concept (i.e., the acquired rule) substitute constants in the explanation structures. This entire process can be viewed as knowledge transformation, i.e., it transforms existing non-operational knowledge into a form that is usable and operational in solving problems. Therefore the task of the explanation-based system is summarized that the learning component of the system learns target concepts in operational form by operationalizing the features of the training example for better problem-solving performance of the system. This results in better problem-solving performance.

Another scheme for learning from examples has been studied in the context of achieving efficiency and accuracy in reasoning by pruning rule sets [11]. The goal of this scheme is to resolve possible inefficiencies and inaccuracies caused by representational biases. For example, decision tree generated by classification algorithm can be converted into a set of rules, one for each path from the root to a leaf in the decision tree. Each rule is then pruned by a significance filtering which identifies the significant attribute-values for classifying the training examples. Pruning spurious and irrelevant attribute-values on the rules is the key to achieving efficient and accurate performance. In addition, further efficiency is achieved during converting the rules back into a tree structure, which makes the search more efficient.

4. Problem Solving Primitives for Efficient Reasoning

The primary method for achieving efficiency has been to adopt problem-solving methods which are most appropriate to the type of the task. The features of a particular problem-solving type in a task framework are incorporated as two factors: kind of knowledge required, and the inferencing strategies used. These constitute the task architecture when designing a knowledge-based system for a particular problem-solving type. The strength of the task methodology comes from the fact that an overall analysis of the requirement for the problem solution determines a set of tasks that provide the building blocks for system design. However, this approach to using high-level building blocks may not be sufficient for knowledge base refinement because task definitions are often too coarse grained and too abstract to incorporate important features and distinctions that arise across different problem domains in the same task category. Too coarse a granularity affects explanation generation and consequently knowledge refinement, because in order to identify and locate the cause of an error, the knowledge refinement system should be able to explain the details of steps performed by the performance system. Also, tasks tend to adopt fixed task structures, which makes it impossible to reconfigure task structures depending on problem-solving context, where a problem-solving context is determined by identifying what the system is doing in a certain situation.

These disadvantages are fixed by introducing a set of finer grained problem-solving primitives, some of which together accomplish the goal of a problem-solving method invoked by a subtask. *Cover and differentiate* PSM for example, may include the following problem-solving primitives: covering, differentiating, evidence combination, ranking, and evidence gathering. The problem-solving primitives are determined by the problem-solving context or state. For example, if some specific information is required at a certain step which corresponds to a problem-solving state, a primitive, evidence gathering, is called upon to collect information from the user. Applications of this primitive to the current state of the system could lead to a new state. Therefore, a framework with problem-solving primitives enables the entire problem-solving to be explicitly represented as a state diagram. State diagrams provide the basis for more detailed explanations of problem-solving behavior, which is then a key to performing better knowledge refinement, when required. This will be discussed in the next section.

A simplified version of the hierarchy for the *Establish and Select* PSM appears in Table 2. This PSM is composed of problem solving constructs (PSCs). Each PSC is composed of problem solving primitives (PSPs). We also discussed several machine learning paradigms based on analysis of problem-solving experience or training examples that assist in improving problem-solving performance. These paradigms have been successful in domains where the domain knowledge is well defined and therefore, the problem can be clearly described by its domain theory. The structured problem-solving traces provide a basis for the analysis of the problem-solving performance and allow the system to acquire better problem-solving knowledge and refine it for efficient problem-solving. However, in real-world applications where the domain is fuzzy and complex, and therefore, problem-solving requires a number of heuristic approaches to solve the problem, it is hard to establish structured problem-solving traces. When we work on fuzzy and complex domains, we need to develop an alternative approach such as conceptual clustering methods to reorganize the problem solving knowledge encoded as rules and improve performance as discussed above.

Table 2. Problem Solving Primitives of PSM

PSM	PSC	PSP
Establish	Get Goal	
	Evidence Seek	Get Info Query Subgoal
	Evidence Cover	Evidence Mark Rule Search
	Evidence Combination	Regular DS Hierarchical DS
	Exit Check	Flat Case Hierarchical Case Move to Parent Goal
Select	Hypothesis Select	Flat Case Hierarchical Case
	PSP Select	Differ Filter Max Hypothesis
	Evidence Select	Rule Model Contribution Measure

5. State Diagram for Problem Solving

We adopt the *Establish and Select* PSM for classificatory problem-solving as the appropriate problem-solving mechanism. Once the individual components of the control structure are defined in terms of PSP's, the overall control algorithm for the reasoning system can be represented in terms of a finite state diagram as shown in Figure 1. Each node in the figure corresponds to a particular state (or partial state) of the system and an active PSP (the PSP that is currently driving the problem-solving). The links represent the particular effects of applying the PSP to the current state of the system. The result is a new state for the system and the selection of a new PSP.

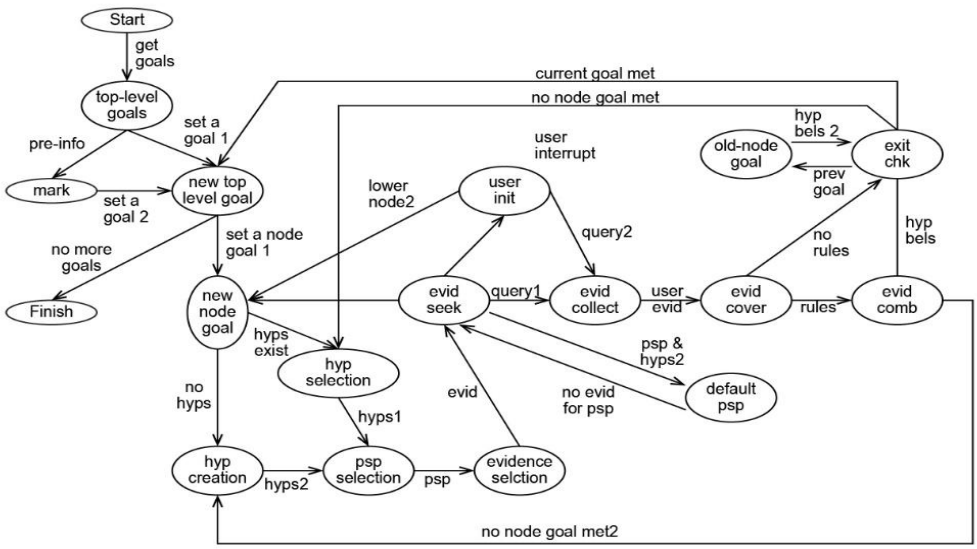

Figure 1. State Transition of Problem Solving Methods

For example, *Evid select* state looks for evidence to support the leading or positive hypotheses in the active hypothesis set and then goes to *Evid seek* state to acquire this evidence. This node (*Evid seek*) has multiple pathways to neighboring nodes. If the evidence selected in *Evid select* is associated with a query, then the control goes to *Evid collect* state. If a subgoal is associated with the evidence, then the next problem-solving state will be *New node goal* state. From the figure it is apparent that the entire procedure is explicitly represented with respect to the states and relevant PSP's. This explicit representation of the process in terms of PSP's is the key factor to explanation generation.

6. Conclusion

This paper presents a methodology for building a knowledge based system equipped with efficient reasoning mechanism. Since the inference mechanism varies significantly from one domain to another, even among domains in same task category, there is a need that the tools should be able to configure the reasoning mechanisms involved in building a knowledge based system to accommodate specific properties of the application domain under consideration. General reasoning schemes are often too coarse to configure the details and particulars of the application system design. This problem is handled with provision of more detailed problem-solving primitives for detailed subtasks. These are configured to explicitly express the overall control structure. This clear and modular definition of subtasks for the given problem and explicit control structures benefit system designers and knowledge engineers in designing and implementing a knowledge-based system. That is, this design framework can allow them to speed up the system design process and make effective in maintaining the knowledge-based system. More importantly, to improve the effectiveness of the reasoning system, the suggested reasoning scheme utilizes explicit problem solving primitives defined in a form of state diagram.

References

[1] R. Greiner, C. Darken, and N.I. Santoso, "Efficient Reasoning," Journal of ACM Computing Surveys, Vol. 33, Iss. 1, pp. 1-30, Mar, 2001.
DOI:10.1145/375360.375363.

[2] A. Avron, B. Konikowska, and A. Zamansky, "Efficient Reasoning with Inconsistent Information using C-Systems," Information Sciences, Vol. 296, No.1, Nov. 2014.
DOI:10.10.16/j.ins.2014.11.003

[3] T. KamiM. Knorr, and J. Leite, "Efficient Paraconsistent Reasoning with Ontologies and Rules," Proc. of the 24th Inter. Joint Conference on Artificial Intelligence, pp. 3098-3105, 2015.

[4] K.R. Thorisson, T. Thorarensen, J. Siguroardottir, and B. Steunebrink, "Why Artificial Intelligence Needs a Task Theory," Atrificial General Intelligence, Vol 9782, pp. 118-128, 2016.

[5] E.K. burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and j. Woodward, "A Classification of hyper-heuristic Approaches," Handbook of metaheuristics, pp. 449-468, Aug. 2010.
DOI:10.1007/978-1-4419-1665-5_15

[6] K. Munir and M. S. Anjum, "The Use of Ontologies for Effective Knowledge Modelling and Information Retrieval," Applied Computing and Informatics, Vol. 14, Issue 2, pp. 116-126, Jul. 2018.
DOI:10.1016/j.aci.2017.07.003

[7] E. Alpaydin, "Machine Learning: The New AI," MIT Press, Sep. 2016.

[8] A Nuxoll and J. Laird, "Enhancing Intelligent Agents with Episodic Memory," Cognitive Systems Research, Vol. 17 pp. 34-48, Jul. 2012.
DOI:10.1016/j.cogsys.2011.10.002

[9] L. Chrpa, "Generation of Macro-operators via Investigation of Action Dependencies in Plan," Planning and Scheduling, Vol. 25, Issue 3, pp. 281-297, Sep. 2010.
DOI:10.1017/S0269888919999159

[10] T.H. Cho, "Embedding Integglient Planning Capability to DEVS Models by Goal Regression Method," Simulation, Vol. 78 Issue 12, pp. 716-730, Dec. 2002.
DOI:10.1177/0037549702078012002

[11] G.S. Lee and I.K. Kim, "A Study on Simplication of Machine Learning Model," Journal of the Institute of Internet, Broadcasting and Communication, Vol. 15, No. 4, pp. 147-152, Aug. 2016, DOI:10.7236/JIIBC.2016.16.5.147