Measuring Evaluation Parameters in Benchmarking Rule Scheduling Methods in Active Database Systems

Abbas Rasoolzadegan, Rohollah Aleshekykh, Ahmad Abdollahzadeh
Intelligent Systems Laboratory
http://ce.aut.ac.ir/islab
Department of Computer Engineering and Information Technology
Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran
{Rasoolzade, Aleshekykh, Ahmad}@aut.ac.ir

Abstract

Active database systems (ADBS) can react to the occurrence of some predefined events automatically. Reactive behavior of ADBS is organized by a collection of active rules. One of the most important modules of ADBS is the rule manager. The main responsibility of the rule manager is triggering, buffering, firing and selecting (scheduling) rules. Rule scheduling method has considerable impact on performance and efficiency of ADBS. In this paper, we measure evaluation parameters in benchmarking rule scheduling methods in ADBS. We first investigate existing rule scheduling methods and then propose a framework to compare and evaluate these methods. In this framework, five evaluation criteria have been proposed: Average Response Time, Response Time Variance, Throughput, Time Overhead per Transaction, and CPU Utilization. Existing methods have been evaluated by using this framework and the method which has the most positive impact on performance and efficiency of ADBS has been selected by analyzing the weaknesses and strengths of existing method. (This method has been named E_c-SJF_PRC_V.1.8). We conclude from results of experiments that using learning techniques in E_c-SJF_PRC_V.1.8 improve the performance and efficiency of Active Database Management System.

1. Introduction

Traditional database systems (passive databases) can only store and retrieve data [1]. They work directly with user requests. By application getting grown and databases getting larger, passive database systems were unable to manage these large systems, and as a result, database systems needed to supervise some predefined special situations and react to some occurrences [2]. These predefined situations are called events. For supporting reactive behavior of new database systems (called Active Database Systems), Event-Condition-Action (ECA) rule format was created. The ECA format has three sections: Event, Condition, and Action. When an event occurs, condition gets evaluated and if the condition is true, action is executed.

1.1. Active Database Management Systems (ADBMS)

There are many types of events in different ADBS, such as data insertion, data manipulation, transaction start, commit, abort and rollback. The conditions specify what should be checked after the occurrence of an event and before execution of an action. The actions can contain anything such as: data modification and retrieval (in relational DBMS), transaction operation like commit or abort, method invocation (in object-oriented DBMS), procedure call (in relational DBMS) and rule operations [3].

One of the most important aspects of the ADBS that affects their power is rule language [4]. Besides the main components of rule language, there are additional features that play an important role in the specification of ECA rules. One of these important features is coupling modes. The phases of rule execution discussed so far are not necessarily executed contiguously, but depend on the so-called coupling modes which are pairs of values (x, y) associated with each rule. The value ‘x’ couples event signaling and condition evaluation of a rule, whereas ‘y’ couples condition evaluation and action execution. Possible
coupling modes are immediate, deferred and independent [4]:

- Immediate mode: in this mode, when an event occurs, current transaction is suspended and the action is executed, if the condition holds.
- Deferred mode: in this mode, after the occurrence of an event condition evaluation and action execution is deferred till the end of the current transaction. In deferred mode, the action of triggered rule should be executed before current transaction commits.
- Independent mode: when an event is triggered in independent mode, there are no time-constraints and restrictions on condition evaluation and action execution.

In order to support reactive behavior of the ADBS, they should contain additional units for managing rule base and rule processing steps in comparing with Passive Database Management systems. Such a DBMS is called an Active Database Management System (ADBMS) [5].

1.2. Rule Processing In Active Database Systems

In this section, we briefly describe what happens when an event occurs. An application runs sequentially until an event occurs. After an event occurs, the rule processing unit is activated and triggers the appropriate rule(s). Triggered rule(s) are queued in a temporary buffer. Then triggered rule(s) are selected according to some special criteria and then their “condition” section is evaluated. If condition is true, the action section will be executed. If the current rule triggers some other rules, new triggered rules will be passed to the rule processing unit. When there aren’t any triggered rules, the application continues running. In summary, there are five different rule processing steps:

(a) Event Signaling: When a primitive event occurs, the primitive event detector signals that event. Additionally, the composite event detector considers these primitive events that contribute to composite events.

(b) Rule Triggering: After the event is signaled, ECA rules that correspond to the signaled event are selected, and for each of them rule instances are created. In each rule instance, there is some additional information based on scheduling method, such as timestamp, deadline, execution time, etc. These rule instances are buffered to use in the next step.

(c) Condition Evaluation: After the buffering of rule instances, their conditions are evaluated.

Then, for each rule with a true condition, a transaction is generated.

(d) Transaction Selection: This step is also called transaction scheduling phase. In this phase, a selection algorithm [6] operates on execution buffer and selects one transaction which is generated based on triggered rules, and sends the transaction to the execution unit.

(e) Transaction Execution: Transactions generated based on triggered rules are executed in this phase.

This paper has four sections. In section two, we analyze existing rule scheduling methods in ADBS. In section three, we introduce a framework to compare and evaluate existing rule scheduling methods. In this framework, five evaluation criteria have been proposed: Average Response Time, Response Time Variance, Throughput, Time Overhead per Transaction and CPU Utilization. Existing methods have been evaluated by using this framework and the method which has the most positive impact on performance and efficiency of ADBS has been selected by analyzing the weaknesses and strengths of existing methods (E.g., SJFPRO-V.1.8). At the end, in section four, there is a conclusion and some favorite jobs we tend to do in the future for improvement of more rule scheduling in ADBS.

2. Rule Scheduling Methods

In this section we briefly describe the methods used for rules scheduling. For selecting one of the buffered rules the execution unit uses a selection algorithm. There are numerous methods for rule scheduling in ADBS [7]. Rule processing in ADBS has an active nature [7]. This means that each triggered rule might cause other rules to fire. So ADBS can not use typical scheduling methods. Figure 1 shows the formal specification of a scheduling method in general.

```
1) RULE_BASE = {Set of ECA RULE}
2) ACTIVE_RULE_BASE = {Set of ECA RULE}
3) Input = RULE_BASE
4) Output = RULE
5) n = |i ∈ N | i = Number of ECA RULES in the RULE_BASE
6) ACTIVATE(R) = {Create Instance of R}
7) ADD_ACTIVE_RB(R) = |i ∈ N and i = n | ACTIVE_RULE_BASE[i+1] = R
8) RULE_SELECTION (ACTIVE_RULE_BASE)
```

Figure 1: The formal specification of scheduling method in general

In the rest of this section we briefly describe rule processing methods used in ADBS. In this paper, we
use “rule selection” and “transaction selection” terms interchangeably.

- **Random Scheduling Method:** Random selection is one of the easiest methods for rule scheduling in ADBS [10]. This method has been implemented in RPL and Ode active database systems [10]. In the Random method ADBMS selects one of the activated rules randomly. The most important characteristic of this method is its simplicity, at the cost of efficiency. The formal specification of the random scheduling method in ADBMS is obtained by replacing line 8 in figure 1 with the following pseudo code.

$$8' \) \text{RANDOM\_GENERATOR}(n) = \{ i \in N \mid 1 \leq i \leq n \text{ and } i \text{ selected randomly} \}$$

$$\text{RULE\_SELECTION( ACTIVE\_RULE\_BASE ) = }$$

$$\{ \forall R_i \in \text{ACTIVE\_RULE\_BASE}( \text{R}\_s, \ldots \text{R}\_n) \}, \text{R}_i \text{ Selected} \mid \{ \forall R_i \in \text{ACTIVE\_RULE\_BASE} \mid 1 \leq i \leq n \}

\{ \forall R_i \in \text{ACTIVE\_RULE\_BASE} \mid 1 \leq i \leq n \}

\{ \forall R_{s, \text{Static Priority} = R_{s, \text{Static Priority}}} \}$$

- **Static Priority Scheduling Method:** In this method, the system assigns a numeric priority to each ECA rule but the priorities need not be unique. In the Ariel [11] and POSTGRES [3] systems, each rule is assigned a priority between -1000 and +1000. When an activated rule should be selected to run, the rule that has the minimum static priority is selected. The formal specification of static priority method in ADBMS is obtained by replacing line 8 in figure 1 with the following pseudo code.

$$8' \) \text{RULE\_SELECTION( ACTIVE\_RULE\_BASE ) = }$$

$$\{ \forall R_i \in \text{ACTIVE\_RULE\_BASE} \}, \text{R}_i \text{ Selected} \mid \{ \forall R_i \in \text{ACTIVE\_RULE\_BASE} \mid 1 \leq i \leq n \}

\{ \forall R_i \in \text{ACTIVE\_RULE\_BASE} \mid 1 \leq i \leq n \}

\{ \forall R_{s, \text{Static Priority} = R_{s, \text{Static Priority}}} \}$$

- **FCFS Scheduling Method:** FCFS (First Come First Serve) scheduling method is one of the classic methods used for rule scheduling in ADBS [6]. When an event occurs and rules are triggered, an instance of each triggered rule is generated. This instance of triggered rule contains a timestamp which shows the time the rule is triggered. When an activated rule should be selected to run, the activated rule that has the earliest timestamp is selected. This scheduling method is used in SAMOS [12]. The formal specification of FCFS method in ADBMS is obtained by replacing lines 6 and 8 in figure 1 with the following pseudo code.

$$6' \) \text{SET\_TIMESTAMP(R) = R's TimeStamp} = \text{Current Time}$$

$$\text{ACTIVATE(R) = } \{ \text{Create Instance of R and SET\_TIMESTAMP(R)} \}$$

$$8' \) \text{RULE\_SELECTION( ACTIVE\_RULE\_BASE ) = }$$

$$\{ \forall R_i \in \text{ACTIVE\_RULE\_BASE} \}, \text{R}_i \text{ Selected} \mid \{ \forall R_i \in \text{ACTIVE\_RULE\_BASE} \mid 1 \leq i \leq n \}

\{ \forall R_{s, \text{Deadline} = R_{s, \text{Deadline}}} \}$$

- **Concurrent Execution Scheduling Method:** HiPAC active database system [6] supports this method. Rule processing in HiPAC is invoked whenever an event occurs and triggers one or more rules. This method differs from most other rule scheduling methods in its handling of multiple triggered rules. So we can not quantitatively compare it with other rule scheduling methods which are serial. HiPAC executes all triggered rules concurrently. This means that if during rule execution, additional rules are triggered, they are executed concurrently. Another ADBMS called FAR [8] also uses concurrent rule execution.

- **EDF based Scheduling Method:** Earliest Deadline First (EDF) is one of the classic algorithms for transaction scheduling in real-time systems [7]. The EDF based method [7] is one of the best methods introduced for rule scheduling till now. This method has been presented for Real-time Active Database (RADB). In this method rules are scheduled based on their deadline. This method has three different versions: (1) EDF1, (2) EDF2, and (3) EDF3. The EDF1 is a static baseline policy where rules priorities do not change with time. EDF2 and EDF3 are dynamic policies where rules priorities change depending on the amount of dynamic work they have generated [7]. The formal specification of EDF based method in ADBMS is obtained by replacing line 8 in figure 1 with the following pseudo code.

$$8' \) \text{RULE\_SELECTION( ACTIVE\_RULE\_BASE ) = }$$

$$\{ \forall R_i \in \text{ACTIVE\_RULE\_BASE} \}, \text{R}_i \text{ Selected} \mid \{ \forall R_i \in \text{ACTIVE\_RULE\_BASE} \mid 1 \leq i \leq n \}

\{ \forall R_{s, \text{Deadline} = R_{s, \text{Deadline}}} \}$$

- **E-c-SJF Scheduling Method:** The SJF algorithm is one of the classic scheduling algorithms. This method is one of the most effective scheduling methods [13]. SJF algorithm is not useful for rule scheduling in ADBS due to active work load nature [7] of it. So there is defined preprocess for preparing rule base to use the SJF algorithm for rule scheduling in Ex-SJF (Extended SJF) method [9]. The difference between SJF and Ex-SJF is in manner of transactions (rules) execution time calculation. In Ex-SJF method, the execution time of each parent transaction (rule) is related to the number of its immediate and deferred
child transactions (rules). The formal specification of 
E_v-SJF method in ADBMS is obtained by replacing line 8 in figure 1 with the following pseudo code.

8') RULE_SELECTION (ACTIVE_RULE_BASE) =
   { ∀ R_i ∈ ACTIVE_RULE_BASE, [(R_i,T_i), ...,(R_n,T_n)], R, Selected } \{ R_i ∈ ACTIVE_RULE_BASE | 1 ≤ i ≤ n \}
   \{ ∀ R_j ∈ ACTIVE_RULE_BASE | 1 ≤ j ≤ n \}
   R_i’s Real Execution Time < R_j’s Real Execution Time \}

According to manner of interference of immediate and
deferred child transactions (rules) execution time in their parent rules execution time, there are two versions of 
Ex-SJF which are named Ex-SJF_{EXA} and 
E_v-SJF_{PRO} [9].

Although E_v-SJF is generally more effective than other
mentioned scheduling methods, it has some
weakness points such as: (1) It is useless in real-time
systems and systems with concurrent execution ability.
(2) It does not calculate the execution time of
transactions, exactly.

• E_v-SJF_{PRO-V.1.8} Scheduling Method: In this
method to solve the second weakness of E_v-SJF_{PRO}, an
Event Triggering Probability Estimation algorithm has
been developed and this algorithm has been integrated
to E_v-SJF_{PRO}. This algorithm helps E_v-SJF_{PRO} in
calculating the exact execution time of rules [15]. The

formal specification of E_v-SJF_{PRO-V.1.8} method is
obtained by replacing pseudo code table 1 instead of
line 8 in figure 1.

3. Proposed Framework to compare and
evaluation of existing rule scheduling
methods

In this section we introduce a framework for comparison and evaluation of existing rule scheduling
methods [14]. This framework contains five evaluation
criteria: Average Response Time, Response Time
Variance, Throughput, Time Overhead per Transaction
and CPU Utilization. Table 2 defines these parameters.

We need an environment which can simulate an active
database system. With such system we can implement
each rule scheduling method and consider the
performance of it. For this reason, an environment
which is named Active Database System Simulator
(ADSS) has been designed and implemented at the
Intelligent Systems Laboratory [9].

Figure 2 illustrates the architecture of the ADSS.
The ADSS has three main modules such as “object
manager unit”, “rule manager unit” and “transaction
manager unit” [14].

Experiments are performed in three modes [9]: (1)
“Deferred mode”, (2) “Immediate mode” and (3)

Table1: Pseudo code of rule execution time calculation and method of rule selection in E_v-SJF_{PRO-V.1.8}

<table>
<thead>
<tr>
<th>Rule (R)</th>
<th>Condition_Probability(R → C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>For Each LS's of R → C That LS → fixed=False Do</td>
</tr>
<tr>
<td>Calc_Execution_Time(R)</td>
<td>LS_Probability(R → C → LS)</td>
</tr>
<tr>
<td>BEGIN</td>
<td>IF R → C → LS → Fixed=False THEN</td>
</tr>
<tr>
<td></td>
<td>R → C → Fixed=False</td>
</tr>
<tr>
<td></td>
<td>R → LS_Probability(R → C → LS)</td>
</tr>
<tr>
<td></td>
<td>END</td>
</tr>
<tr>
<td></td>
<td>END</td>
</tr>
<tr>
<td>END</td>
<td>END</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule (R)</th>
<th>RULE_SELECTION(ACTIVE_RULE_BASE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>{ ∀ R_i ∈ ACTIVE_RULE_BASE, [(R_i,T_i), ...,(R_n,T_n)], R, Selected</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Pseudo code of rule execution time calculation and method of rule selection in E_v-SJF_{PRO-V.1.8}
“Composite mode”. In the first mode system uses rules only in deferred mode. In the second mode, system uses rules only in immediate mode and ultimately in the third mode, system uses rules in all immediate, deferred, and independent modes. Results of experiments in deferred, immediate and composite modes are shown in tables 3, 4, 5, respectively. The content of each cell shows the rank of corresponding scheduling method according to corresponding evaluation criteria.

**Table 2: Definition of evaluation parameters**

<table>
<thead>
<tr>
<th>N = Number of Executed Rules</th>
<th>ART = Average Response Time</th>
<th>RTSV = Response Time Standard Variance</th>
<th>UCPU = CPU Utilization</th>
<th>TOPT = Time Overhead Per Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_i = Activation Time of i-th Rule</td>
<td>T_i = Start of Execution Time of i-th Rule</td>
<td>T = (T_i + Execution Time of N-th Rule) - T_i</td>
<td>T = \sum_{i=1}^{N} Real Execution Time of i-th Rule</td>
<td>U_CPU = \frac{T - T}{T} * 100</td>
</tr>
</tbody>
</table>

**Table 3: Results of simulation of available rule scheduling methods in deferred mode**

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Methods</th>
<th>Average Response Time</th>
<th>Response Time Variance</th>
<th>Throughput</th>
<th>Transaction Overhead</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Random</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Static Priority</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>FCFS</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>EDF</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>EDFpiv</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Ec-SJFEXA</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Ec-SJFPRD</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Ec-SJFPRD.V.1.8</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Results of experiments show that Ec-SJFPRD.V.1.8 has generally the most positive impact on performance (Response Time, Response Time Variance, and Throughput) and efficiency (Time Overhead per Transaction and CPU Utilization) of ADBS [15].

**Table 4: Results of simulation of available rule scheduling methods in immediate mode**

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Methods</th>
<th>Average Response Time</th>
<th>Response Time Variance</th>
<th>Throughput</th>
<th>Transaction Overhead</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Random</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Static Priority</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>FCFS</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>EDF</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>EDFpiv</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Ec-SJFEXA</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Ec-SJFPRD</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Ec-SJFPRD.V.1.8</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table 5: Results of simulation of available rule scheduling methods in composite mode**

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Methods</th>
<th>Average Response Time</th>
<th>Response Time Variance</th>
<th>Throughput</th>
<th>Transaction Overhead</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Random</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Static Priority</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>FCFS</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>EDF</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>EDFpiv</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Ec-SJFEXA</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Ec-SJFPRD</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Ec-SJFPRD.V.1.8</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6 shows the percentage of optimizing the rule scheduling in Ec-SJFPRD.V.1.8 in compare with Ec-SJFPRD based on three evaluated parameters: average response time, response time variance and throughput.

**Table 6: Percentage of rule scheduling optimization in Ec-SJFPRD.V.1.8 in compare with Ec-SJFPRD [15]**

<table>
<thead>
<tr>
<th>Evaluation parameter</th>
<th>Mode</th>
<th>Average Response Time</th>
<th>Response Time Variance</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>8%</td>
<td>12.6%</td>
<td>9%</td>
<td></td>
</tr>
<tr>
<td>Deferred</td>
<td>18.8%</td>
<td>33.6%</td>
<td>14.3%</td>
<td></td>
</tr>
<tr>
<td>Composite</td>
<td>14.2%</td>
<td>21.4%</td>
<td>12%</td>
<td></td>
</tr>
</tbody>
</table>

Results of experiments show that by adding estimation module to Ec-SJFPRD, executing time of rules are calculated more exactly (the amount of this
precision depends on value $\epsilon$) and leads to optimizing the Average Response Time, Response Time Variance and Throughput of $E_\epsilon$-SJF\textsubscript{PRO} approach. Event triggering probability estimation process does not impose any overhead on the ADBS. So the Time Overhead per Transaction and CPU Utilization of $E_\epsilon$-SJF\textsubscript{PRO}-V.1.8 and $E_\epsilon$-SJF\textsubscript{PRO} are equal.

4. Conclusions and future works

In this paper, we first analyze existing rule scheduling methods in ADBS. Then we introduce a framework to compare and evaluate existing rule scheduling methods. In this framework, five evaluation criteria have been proposed: Average Response Time, Response Time Variance, Throughput, Time Overhead per Transaction and CPU Utilization. Existing methods have been evaluated by using this framework and the method which has the most positive impact on performance and efficiency of ADBS has been selected by analyzing the weaknesses and strengths of existing methods ($E_\epsilon$-SJF\textsubscript{PRO}-V.1.8).

In future, we intend to increase the efficiency of $E_\epsilon$-SJF\textsubscript{PRO}-V.1.8 method by adding a learning module (using techniques such as Learning Automata and Reinforcement). Learning techniques cause to calculate the value $\epsilon$ automatically. System calculates the value $\epsilon$ according to feedback which is received from functionality of the system. Functionality of the system is determined based on system’s status according to evaluated criteria.

Acknowledgment

The authors would like to thank Mr. Mohammad Reza A. Shirazi for his technical assistance and helpful advices.

References