Search for Minimal and Semi-Minimal Rule Sets in Incremental Learning of Context-Free and Definite Clause Grammars

Keita IMADA\(^{1(a)}\), Nonmember and Katsuhiro NAKAMURA\(^{1(b)}\), Member

SUMMARY This paper describes recent improvements to Synapse system for incremental learning of general context-free grammars (CFGs) and definite clause grammars (DCGs) from positive and negative sample strings. An important feature of our approach is incremental learning, which is realized by a rule generation mechanism called "bridging" based on bottom-up parsing for positive samples and the search for rule sets. The sizes of rule sets and the computation time depend on the search strategies. In addition to the global search for synthesizing minimal rule sets and serial search, another method for synthesizing semi-optimum rule sets, we incorporate beam search to the system for synthesizing semi-minimal rule sets. The paper shows several experimental results on learning CFGs and DCGs, and we analyze the sizes of rule sets and the computation time.

key words: grammatical inference, CFG, DCG, beam search, Synapse

1. Introduction

Grammatical inference, or learning grammars from positive and negative samples, is a fundamental research subject in machine learning. Similar to other machine learning subjects, the most important problem in learning grammars is the degree of computational complexity. Some theoretical studies showed that context-free grammars (CFGs) and more restricted grammars are not learnable in polynomial time\(^{[15]}\). This is a reason that many studies on learning CFGs\(^{[3]}\), \(^{[17]}\) restrict the target grammars to subclasses of CFG or add some structural information to the samples for polynomial-time learning.

There are several approaches for solving hard problems in information science that require non-polynomial time, including approximation algorithms, heuristics and genetic algorithm. Most of these methods aim to find practical or semi-optimum solutions for the problems.

We have been working on incremental learning of general, ambiguous and unambiguous, CFGs and definite clause grammars (DCGs), which are implemented in Synapse system\(^{[10]}\)–\(^{[12]}\). In this paper, we describe recent improvements to Synapse and discuss relationships between the sizes of rule sets and the computation time required for learning the grammars. We applied the inference of DCG in Synapse to learning of a grammar and a compiler of a simple programming language from sample source programs and the corresponding object codes in an intermediate lan-

*This algorithm is described in \(^{[2]}\) as follows: If the learner can successfully analyze a given sentence, the learner’s hypothesis on the target grammar is left unchanged. Otherwise, the learner uniformly selects a parameter and changes its value and tries to reprocess the sentence.
studies on learning minimal or semi-minimal rule sets of CFGs or higher grammars. On the other hand, many works on regular grammars are related to minimal state sets. There have been studies for learning complex CFGs from large volumes of samples, such as the Omphalos competition for learning CFGs [18].

The DCG is an extended CFG based on logic programming [14]. DCGs are more powerful than CFGs because the rules in DCGs can have additional arguments for controlling the derivation and for communicating information between the nodes in the derivation trees. Learning DCGs is closely related to inductive logic programming (ILP) [13]. Several studies have dealt with learning grammars based on ILP [9]. Cussens and Pulman [1] described a method of learning missing linguistic rules using bottom-up ILP. However, there have been few publications focusing on learning DCGs. Fredouille et al. [4] described a method for efficient ILP-based learning on DCGs for biological pattern recognition. Ross [16] presented learning of Definite Clause Transduction Grammars, a logical version of attribute grammars, closely related to inductive logic programming (ILP) [13].

Several studies have dealt with learning grammars based on ILP [9]. Cussens and Pulman [1] described a method of learning missing linguistic rules using bottom-up ILP. How-
ever, there have been few publications focusing on learning DCGs. Fredouille et al. [4] described a method for efficient ILP-based learning on DCGs for biological pattern recognition. Ross [16] presented learning of Definite Clause Transduction Grammars, a logical version of attribute grammars, by genetic programming.

This paper is organized as follows. Section 2 briefly describes definitions of DCGs and their derivations, which includes those of CFGs. Section 3 describes the bridging rule generation procedure for DCGs with an example. Section 4 shows three search strategies: global search, serial search and beam search. Section 5 shows experimental results of learning several CFGs and DCGs, and compares computation times and sizes of rule sets synthesized by the three search strategies. Finally, Sect. 6 concludes and describes future subjects.

### 2. Context-Free Grammars and Definite Clause Grammars

We use the notations and syntax of standard Prolog for variables, terms and operators. The reader can refer to a textbook [5] for definitions in formal languages and to a textbook, for example [8], for definitions in logic programming such as substitutions, instances, most general unifiers (mgu), and renaming of variables.

For any terms \( s \) and \( t \), we write \( s \geq t \) and say that \( s \) is more general than \( t \), if and only if \( t \) is an instance of \( s \), i.e., there is a substitution \( \theta \) with \( s\theta = t \). We write \( s \equiv_\theta t \), if and only if \( s \) is unifiable with \( t \) by the mgu \( \theta \). For any terms \( s \) and \( t \), \( u \) is the least general generalization (lgg), if and only if \( u \geq s \), \( u \geq t \) and there is no other term \( v \) such that \( v \geq s \), \( v \geq t \), \( u \geq v \) and \( v \) is not a variant of \( u \).

A grammar rule (or simply a rule) is of the form \( A \rightarrow B_1, \ldots, B_n \) \((n \geq 1)\), where \( A \) is a nonterminal term, and each of \( B_i \) is either a terminal symbol or a nonterminal symbol. The nonterminal term is either a nonterminal symbol \( p \) or a complex term \( p(T) \) of a nonterminal symbol \( p \) and a term \( T \) called a DCG term or an additional term \(^7\). Note that we restrict nonterminal terms to have at most one DCG term, or an additional term. A definite clause grammar (DCG) is defined by a pair \( G = (RS, S) \) of a set \( RS \) of grammar rules and a nonterminal term \( S \) called a starting term such that the set of terminal symbols is disjoint with that of nonterminal symbols.

A context-free grammar (CFG) is a DCG such that all the nonterminal terms are nonterminal symbols, or equivalently all the rules have no DCG term. In this paper, we represent terminal symbols by \( a, b, c \), nonterminal terms by \( A, B, C \), nonterminal symbols in CFG by \( p, q, r, \cdots \) and terminal symbols or nonterminal terms by \( \alpha, \beta, \gamma \), unless otherwise specified.

The derivation relation \( \Rightarrow_G \) of a DCG \( G = (RS, S) \) is defined by the resolution for two Horn clauses: for a sequence of terms \( A_1, \ldots, A_m \),

\[
A_1, \ldots, A_{j-1}, \beta_1, \ldots, \beta_k, A_{j+1}, \ldots, A_m \theta
\]

if and only if there is a rule \( R \in RS \), a nonterminal term \( A_j \) with \( 1 \leq j \leq m \), and an mgu \( \theta \) such that \( B \rightarrow \beta_1, \ldots, \beta_k \) is a renamed rule of \( R \) and \( A_j \equiv_\theta B \). The relation \( \Rightarrow_G^* \) is the transitive closure of \( \Rightarrow_G \). The language of a DCG \( G \) is the set of sequences \( W \) of terminal symbols with \( S \Rightarrow_G^* W \). Chomsky normal form (CNF) rules are of the form \( A \rightarrow a \) or \( A \rightarrow BC \). Revised CNF rules are of the form \( A \rightarrow \beta \) or \( A \rightarrow \beta y \).

**Example 1:** A DCG for Fibonacci numbers

The pair \( (RS, fib(X)) \) is a DCG for the set of unary Fibonacci numbers, where \( RS \) is the set of the following grammar rules.

\[
\begin{align*}
\text{fib}(1) & \rightarrow a. \\
\text{fib}(t(1)) & \rightarrow a. \\
\text{fib}(t(t(N))) & \rightarrow \text{fib}(N), \text{fib}(t(N)).
\end{align*}
\]

In this rule set, the term \( t(0) \) represents the successor of \( 0 \). For example \( t(t(1)) \) is interpreted as the number 3. This DCG derives strings \( a, aa, aaa, aaaaaa, \cdots \).

### 3. Rule Generation by Bridging

The rule generation procedure uses a revised CYK algorithm for bottom-up parsing of the input string using the generated rules. If parsing fails, then the bridging process generates rules in extended CNF, which bridge any missing parts of the incomplete derivation tree.

Figure 1 shows the rule generation procedure, which is a revision of that in [6]. This nondeterministic procedure receives a string \( w = a_1 \cdots a_n \) of terminal terms, a starting term \( S \), a set \( S_N \) of negative samples and a set \( RS \) of rules from the top-level search procedure, and returns an ambiguous or unambiguous grammar in extended CNF, which derives the string \( w \) but no string in \( S_N \) from \( S \).

\(^7\)This form of the grammar rule of DCG is different from the common form in Prolog in that the terminal symbol is of the form \([a] \).
Example 2: Generation of CFG Rules for Balanced Parentheses

Consider learning of an unambiguous CFG of the balanced parentheses language: that is, the set of strings composed of equal numbers of a’s and b’s such that every prefix does not have more b’s than a’s. For the positive sample ab, Op. 2 in the rule generation procedure synthesizes rule $s \rightarrow ab$, and for $abab$ and $aabb$, Ops. 5 and 2 synthesize rules $s \rightarrow ap$ and $p \rightarrow bs$ | $sb$. After the system generates these rules, it produces the incomplete derivation tree shown in Fig. 2 (a) for the sample $aabbabab$. The system searches for the gap by Op. 4 using rule $s \rightarrow ap$ as shown by the arrows in Fig. 2 (b), and synthesizes the two rules $p \rightarrow sq$ and $q \rightarrow bs$, which have a new nonterminal symbol q, and bridges the gap shown by the thick lines in the figure by Ops. 5 and 2.

4. Searching for Rule Sets

Synapse takes as input ordered sets $S_P$ of samples and $S_N$ of negative samples, and a set $RS_0$ of optional initial rules for incremental learning of the grammar. The samples for learning a CFG are strings, whereas those for learning a DCG are pairs of strings and instances of the starting term. The system searches for any set $RS$ of rules with $RS_0 \subseteq RS$ that derives all the strings in $S_P$ but no negative samples.

The system has three search strategies: global, serial, and beam search. We can represent the search strategies by a search tree for a sequence of positive samples such that:

1. The root node is labeled the initial rule set; and
2. Each node at depth $j$ is labeled a rule set $RS$ such that $RS$ is a superset of the rule set of its parent node and generated for the first $j$ positive samples, and hence $RS$ derives all these positive samples but no negative samples.

The combination of the rule generation procedure and the global search is correct in the sense that every output rule set derives all positive sample strings, and none of the negative samples. Nakamura [12] showed that the bridging rule generation also has the “completeness” for finding minimal CFGs in the revised CNF for any given.
samples and sets of initial rules. It has not been proved that this property also holds for synthesizing grammars in CNF and extended CNF.

4.1 Global Search

In global search, the system scans every node in the search tree within a certain depth to find a minimal set of rules by using iterative deepening with the number of rules. Figure 3 shows the top-level global search procedure. The system controls the search by iterative deepening on the depth, which is the number \( K \) of rules. This control ensures that the procedure finds a grammar with a minimal number of rules at the cost that the system repeats the same search each time the limit is increased.

4.2 Serial Search

In serial search, the system traces one route in the search tree such that each branch to the child node represents a minimal increment of the number of rules. This search is a greedy algorithm in the sense that the system finds a minimal set for each step of the process. This method is also an incremental learning algorithm because each step learns just one positive sample. The cost of the serial search is the sum of the costs of partial searches for a minimal rule set for each positive sample.

Figure 4 shows the top-level serial search procedure, in which the system generates additional rules for each positive sample by iterative deepening. After the system finds a rule set satisfying a positive sample and no negative samples, the process does not backtrack to redo the search on the previous samples.

In serial search, the system needs to avoid generating any rule that has an instance of the starting term on the right hand side, so that all the rules generated for a new positive sample can be independent from the rules for the previous samples. For example, consider learning of a CFG for the language \( \{anbn | n \geq 1\} \cup \{bnan | n \geq 1\} \). Once the system has the rule \( s \rightarrow ap \) and \( p \rightarrow sb \) with the starting symbol \( s \), it cannot complete a CFG for this language. Hence, we constrain the system not to generate rules of the form either \( A \rightarrow s\beta \) or \( A \rightarrow \beta s \) containing the starting symbol \( s \) in the right hand side.

4.3 Beam Search

For beam search, the system has a queue that stores a number of nodes in the search tree. It selects the best node by evaluating the nodes and adds the child nodes extended from the selected node to the queue. Note that serial search is a special case of beam search such that the queue holds only one node and the system always selects the first extended node.

Figure 5 shows the top-level beam search procedure. The evaluation value is the ratio \( |RS_j|/(p+cq) \) of the number of the current rules to an estimated number of samples that the rule set derives, where:

- \( p \) is the number of positive samples including a predetermined number (usually 10 to 50) of the remaining positive samples that the rule set parses. For counting this number, the system parses the remaining positive samples;
1. \( c < 1 \) is a coefficient, which we usually set to 0.2 to 0.4; and
2. \( q \) is the number of remaining positive samples that are 
   “semi-parsed” by the rule set: that is, the samples can 
   be derived by the rule set with an additional rule. 
The system finds this value by applying the \( \text{GenerateRule} \) 
procedure to a predetermined number of the remaining 
positive samples.

This value is intended to represent the distance from the 
node to a preferable goal node. The value generally de-
creases as learning proceeds, and converges to the ratio of 
the number of rules to the number of positive samples, when 
learning succeeds.

**Example 3: A Search Tree of Learning a CFG**

The search tree in Fig. 6 shows how the system synthesizes 
an unambiguous grammar for the balanced parentheses lan-
guage by beam search with the queue length \( m = 10 \). 
Following samples and no initial rules are given for this search.

**Positive samples:** \( ab, aabb, abab, aababb, aababa, 
aababbb, aababab, aabbabab, aabbaabbab, 
aababbbabb. \)

**Negative samples:** \( aab, abb, aba, abbba, aaaaa, bbbb, 
aab, abba, aabba, aabab. \)

Among 11 positive samples, the system uses first four samples 
for the rule generation as shown in the figure, and other 
positive samples for evaluation and checking the grammar. 
The nodes are increasingly ordered according to the evaluation 
values \( |RS|/(p + 0.3 \cdot q) \) in the boxes. The numbers 
(1) to (6) denote the order in which the system extends the 
nodes. The rule set in node (6) satisfies all the positive and 
negative samples.

5. **Experimental Results**

This section shows experimental results obtained by 
Synapse Version 4, written in Prolog, using a Xeon proces-
sor with a 3.6 GHz clock and SWI-Prolog for GNU/Linux. 
We checked the correctness of several synthesized gram-
mars manually and by making the grammars enumerate 
strings. For the other synthesized grammars, we only tested 
their correctness and ambiguity for a large number of sam-
ple. We use the number \( GR \) of all generated rules as an 
index of the size of the search tree, which does not depend 
on the number of samples and the processor environment 
as the computation time. In general, the number of sam-
ple does not strongly affect computation time, provided that 
sufficient samples are given to the system. When the system 
succeeds in learning a grammar after using a number of pos-
tive samples for generating rules, it uses the remainder of 
the samples for checking the grammar.

5.1 Comparison of Search Strategies by Randomly Gener-
ated CFGs

To investigate the effectiveness of three search strategies for 
general CFGs, we make the system synthesize CFGs from 
samples of randomly generated CFGs. The procedure for 
this experiment is as follows:

1. Randomly generate 1000 CFGs in CNF on the alphabet 
   \( \{a, b\} \) with 5 to 10 rules.
2. For each generated CFG \( G \), classify all the strings of 
   lengths less than or equal to 10 into the class of positive 
   samples that are parsed by \( G \) and the class of negative 
   samples of the remaining strings.
3. For each pair of positive sample set and negative sam-
   ple set, make Synapse synthesize CFGs by three search 
   strategies.

The results of this experiment are summarized as follows.

- For each sample, Synapse synthesized a CFG by global 
  search such that the rule set is not larger than that of the 
  original CFG. The computation time ranges from 1 to 
  several thousand seconds.
- The system synthesized the CFGs for most of the sam-
  ple sets by the serial search. The sizes of the rule sets in 
  43.4% of these CFGs are the same as those synthesized 
  by the global search. The sizes in 85.4% of the CFGs 
  are no more than three rules greater than those of the 
  corresponding CFG synthesized by global search, and 
  no more than six rules greater for 94.5% of these CFGs.
- For three sample sets among the 1000 sets, learning of 
  CFGs by serial search did not converge to a fixed rule 
  set. Synapse synthesizes CFGs for these sample sets 
  by beam search.

Figure 7 shows the relationship between computation 
times and the sizes of rule sets for learning CFGs by three 
search strategies. For this plot, twenty samples are randomly 
chosen among the samples of randomly generated CFGs.
Fig. 7  Times and the numbers of rules in learning CFGs for randomly generated CFLs.

5.2 Learning CFG Results

Figure 8 shows the computation time and the sizes of rule sets for learning several CFGs for fundamental context-free languages (CFLs) including (a) balanced parentheses, (b) palindromes, (c) regular expressions, (d) strings containing twice as many $b$’s as $a$’s, (e) strings composed of the same number of $a$’s and $b$’s, (f) strings containing more $b$’s than $a$’s, (g) the complement set of (a), and (h) the complement of $\{a^n b^c | n \geq 1\}$. The lines connect the points of three equivalent grammars synthesized for the same lan-

<table>
<thead>
<tr>
<th>Rules</th>
<th>Positive samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Fibonacci numbers</td>
<td>Global search: $#RS = 3$, $T = 0.95$, $GR = 254$.</td>
</tr>
<tr>
<td>$s(1) \rightarrow a$.</td>
<td>[a] - $s(1)$.</td>
</tr>
<tr>
<td>$s(t(1)) \rightarrow a$.</td>
<td>[a] - $s(t(1))$.</td>
</tr>
<tr>
<td>$s(t(t(X))) \rightarrow s(X)$, $s(t(X))$.</td>
<td>[a a a a] - $s(t(t(1)))$.</td>
</tr>
<tr>
<td>(b) Composite (non-prime) numbers</td>
<td>Global search: $#RS = 2$, $T = 195$, $GR = 6782$.</td>
</tr>
<tr>
<td>$p(1) \rightarrow a$.</td>
<td>$a a a a a a a a a - s(t(1))$.</td>
</tr>
<tr>
<td>$q(t(1)) \rightarrow q(X)$, $p(1)$.</td>
<td>$a a a a a a a a a - s(t(1))$.</td>
</tr>
<tr>
<td>$s(t(X)) \rightarrow q(t(X))$, $s(t(X))$.</td>
<td>[a a a a a a a a a] - $s(t(t(1)))$.</td>
</tr>
<tr>
<td>(c) ${a^n b^c</td>
<td>n \geq 1}$</td>
</tr>
<tr>
<td>$e(1) \rightarrow a$.</td>
<td>$[a a a a a a a a a a - s(t(1))].</td>
</tr>
<tr>
<td>$g(1) \rightarrow c$.</td>
<td>[a a a a a a a a a a].</td>
</tr>
<tr>
<td>$e(t(X)) \rightarrow e(1), s(X)$.</td>
<td>$[a a a a a a a a a a - s(t(t(1)))].</td>
</tr>
<tr>
<td>$p(X) \rightarrow f(1), g(X)$.</td>
<td>$[a a a a a a a a a a - s(t(t(t(1))))].</td>
</tr>
<tr>
<td>(d) Simple English grammar ($s$: singular, $p$: plural)</td>
<td>Serial search: $#RS = 11$, $T = 1.1$, $GR = 2094$.</td>
</tr>
<tr>
<td>article(X) \rightarrow the.</td>
<td>[be has pens] - $s(a)$.</td>
</tr>
<tr>
<td>verb(p) \rightarrow have.</td>
<td>[he has a pen] - $s(s)$.</td>
</tr>
<tr>
<td>be(p) \rightarrow are.</td>
<td>[they have a pen] - $s(p)$.</td>
</tr>
<tr>
<td>noun(s) \rightarrow pen.</td>
<td>[this is a pen] - $s(s)$.</td>
</tr>
<tr>
<td>pronoun(s) \rightarrow this.</td>
<td>[this is the pen] - $s(s)$.</td>
</tr>
<tr>
<td>pronoun(p) \rightarrow they.</td>
<td>[they have the pens] - $s(p)$.</td>
</tr>
<tr>
<td>these are pens] - $s(p)$.</td>
<td></td>
</tr>
<tr>
<td>this is a black pen] - $s(s)$.</td>
<td></td>
</tr>
<tr>
<td>these are black pens] - $s(p)$</td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td>Synthesized DCGs. (Rules with * are given to the system as the initial rules.)</td>
</tr>
</tbody>
</table>

$#RS$: The number of synthesized rules.  $T$: Computation time in seconds.  $GR$: The number of all generated rules.
guage by the three search strategies. All the CFGs are obtained by not restricting the rule generation to unambiguous grammars. Learning unambiguous CFGs generally needs 0 to 100% more rules and longer computation time than those of learning ambiguous grammars for the same languages [12].

These results show that the computation time by serial search is generally much less than that of global search at the expense of producing rule sets that are 1.0 to 3 times larger in most cases. The time and the sizes by beam search are generally medium between those by global and serial searches.

5.3 Learning DCGs Results

As mentioned in Sect. 1, Synapse learned a DCG containing a CFG and translation rules of a compiler of a simple programming language from the sample pairs of source programs and the corresponding object codes [6]. The system synthesized several rules of this DCG representing a syntax-directed translation schema (SDTS) by global search. The learning of DCGs of this type is advantage over learning general DCGs, as the rules have a fixed form for the SDTS.

Figure 9 shows other synthesized DCGs, the rules, positive samples and performance data. The samples are pairs of strings and ground (variable-free) instances of starting terms, except that some negative samples are pairs of strings and the starting term s(X) with a variable X.

Synapse learned DCGs for the following languages by two or more search strategies: (a) the set of unary Fibonacci numbers (Example 1 in Sect. 2) and (b) the set of unary composite (non-prime) numbers. The term s(T) represents a number divisible by T. On the other hand, Synapse learned DCGs for the language (c) \( \{a^n b^n c^n \mid n \geq 1\} \) only by global search and for (d) simple English sentences only by serial search. A reason why the system cannot synthesize some DCGs by serial and beam search is that searching for a non-minimal rule set generally requires a large volume of samples. The system cannot synthesize a DCG for (d) by global search in a reasonable amount of time, because this DCG needs more than ten rules. This problem can be solved by dividing the positive samples into subgroups and using incremental learning.

6. Concluding Remarks and Future Subjects

In this paper, we described methods for incremental learning of minimal and semi-minimal general CFGs and DCGs and showed some experimental results on the relationships between sizes of rule sets and computation time. The experimental results can be summarized as follows.

- Learning CFGs and DCGs by serial search is generally faster than the other two search strategies. Even by this search, the experimental results suggest that computation time grows exponentially with the sizes of rule sets. For a few grammars the learning does not converge to any fixed rule set by this search.

- Beam search is generally more effective for learning the broad class of CFGs than the other two search strategies. A disadvantage is that the parameters and the coefficient might be determined experimentally.

An approach to solve the computation time problem is incremental learning: we can divide the positive samples into subgroups and make the system learn the samples in the subgroups in order. We expect that it is possible to effectively learn grammars of complex languages by incremental learning with appropriate partitioning of the positive samples and ordering the subgroups.

Some languages require a large volume of negative samples in learning their grammars (CFGs and DCGs). We recently improved Synapse so that the negative samples can be given by patterns of strings containing Prolog variables. This is considerably effective to decrease the number of negative samples.

We are applying the learning method for DCG to syntactic pattern recognition of outline figures. The input is a sequence (a string) of terminal terms, each of which represents a primitive of the figures. The recognition process can use the feature of bridging rule generation to find any missing parts not only for learning patterns but also for recognizing patterns from input data with errors or incomplete data.

In addition to this work, we are currently working on another approach to learning CFGs from samples based on the Boolean satisfiability problem (SAT) [7]. In this approach, we represent a set of sample strings and the limits on the size of the rule set of the target grammar as Boolean expressions. The grammar is obtained by using a SAT solver. This method also makes it possible to synthesize minimal rule sets of CFGs only from positive and negative samples. At this moment, some experimental results show that this SAT based method can synthesize most of the fundamental ambiguous CFGs that Synapse has synthesized from samples. There are two fundamental advantages of Synapse over the SAT-based approach: Synapse is based on the incremental learning; and Synapse can synthesize both ambiguous and unambiguous CFGs.

Other important future subjects include: applying our approaches to the methods of ILP; conversely applying ILP methods to DCGs; and applications to learning cellular automata for parallel recognition of languages and two-dimensional patterns.

Acknowledgments

The authors would like to thank Tomohiro Yamada and Hiroyuki Inoue for their help in writing and testing the Synapse system. This work is partially supported by KAKENHI 21500148 and the Research Institute for Technology of Tokyo Denki University, Q09J-06.

References


Keita Imada graduated from the Dept. of Computers and Systems Eng. of Tokyo Denki University in 2006. He is currently a student at the Graduate School of Advanced Science and Technology of the same university. His research interests include machine learning and logic programming. He is a student member of IPSJ.

Katsuhiko Nakamura graduated from the Dept. of Electronic Eng. of Tokyo Denki University in 1966, and received Dr. Eng. degree from the same university in 1980. He is presently a professor of the School of Science and Engineering, Tokyo Denki University. His research interest includes grammatical inference, cellular automata, logic programming and computer Go. Dr. Nakamura is a member of IPSJ, IEEE and ACM.

Keita Imada

Katsuhiko Nakamura