Branch and Bound Algorithm for Phrase Level Pattern Matching By Using Deterministic Context-Free Grammar

Takahisa Kimura*

We have been developing a personal speech understanding system by using a pattern maching method. In this paper, we deal with a phrase-level pattern matching algorithm where LR(1)-syntax is processed. We propose a pairwise Markov model (PMM) which is a system to compute likelyhood of an interpretation for an input speech. A PMM has the same syntactic structure as the finite state sequential decision process (**fsdp**), and is characterized by its base-automaton which is composed through a Cartesian product of two automata. We derive a recurrence equation of a finite state PMM which represents matching of patterns in normal-syntax. We also derive a pattern matching procedure which processes a higher-than-the-normal syntax by using an infinite state PMM (a full PMM). Because of the syntactic equality between a full PMM and an **fsdp**, we apply the Ibaraki's construction method for an **fsdp** extensively so that we obtain the pattern matching procedure as a branch-and-bound procedure (named B & B). We construct a pattern matching algotithm which processes an LR(1)-syntax by using the procedure B & B and a given LR(1) parser.

1. Introduction

For installing speech-perception abilities into machines, many researchers have been investigating continuous speech recognition [1, 2, 3, 4] and speech understanding [5, 6, 7]. We can find recent advances in these fields in [8]; some speaker-independent continuous-word recognizers for large vocabularies are reported. They seem to achieve an automatic transcription of speeches, *i.e.*, an automatic conversion of speeches into its character-representations.

In this research, we have investigated a personal speech understanding system (personal SUS) whose typical task is a voice-control of a personal computer (or a work station) rather than the voice-transcription. We have assumed that almost all sentences in the task obey a predetermined set of syntactic and semantic rules, and that the task required of the personal SUS is a correct extraction of syntactic and semantic information from input speeches. For developing the personal SUS, we have enhanced a dynamic programming procedure which is used for phrase-level pattern matching [9].

For modeling the pattern matching, we introduce a pairwise Markov model (PMM) which is a system to compute likelyhood of an interpretation for an input speech. PMM consists of a composite automaton (**ca**), a cost function associated with each state transition of the **ca**, and an initial cost. A **ca** of the PMM is constructed through a Cartesian product of two automata: one represents a connectivity of speech-segments which are taken from the input speech; the other represents order of reference-words which is identified with syntax of input speeches. Syntactic (and further semantic) information is extracted from the optimum sequence of states of the **ca**.

In order to represent the pa-ttern matching by PMM's, we use an infinite state automaton (**ifsa**) originally proposed by Ginsburg and Glushkov [15] as a **ca**. The Ginsburg-Glushkov **ifsa** (G & G-**ifsa**) M has the Markovian feature (*i.e.*, $M=_{def.}(Q, I, q_0, T, Q_F)$). Hence, similar to the **fsdp** [10, 11], we can apply notions of the dynamic programming to the PMM's. In addition, for any syntax, there exists a G & G-**ifsa** which represents it. Hence, syntax of common programming languages (note that they are in higher-than-the-normal syntax) can be represented by the PMM's. G & G-**ifsa** is descriminated from the **fsa** by its infinite set of states, and G & G-**ifsa** of a finite set of states is reduced to an ordinary **fsa**. Hence, we can represent the PMM as an infinite state sequential decision process (**isdp**) whose base-automaton is composed by two **ifsa**'s through a Cartesian product.

Concerning the recognition algorithm, we consider both finite and infinite state PMM in this paper (called a restricted PMM and a full PMM respectively).

Restricted PMM is a finite state sequential decision process (**fsdp**). Hence, we can solve the optimization problem represented by the restricted PMM through applying a dynamic programming method proposed by Karp and Held [10] and Ibaraki [11]. By using Ibaraki's

*International Institute for Advanced Study of Social Information Science, FUJITSU Ltd.
140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan.

method, we derive a recurrence equation of a restricted PMM. The recurrence equation then derived is equivalent to the Sakoe's recurrence equation of the G2DP algorithm (the generalized two-level DP-matching algorithm [1]), and this result shows the effectiveness of the restricted PMM for representing a continuous-word recognition.

Full PMM is an **isdp**, and it can represent higher-than-the-normal syntax. Because of the syntactic equality between **fsa** and **ifsa**, we apply extensively Ibaraki's construction method for an **fsdp** [12] so that we obtain the pattern matching procedure (named B & B) as form of the branch-and-bound procedure.

For an LR(1)-syntax (*i.e.*, the syntax which is specified by a deterministic context-free grammar [13]), we construct a pattern matching algorithm by using the procedure B & B through the following steps. We find heuristically a conversion method of an LR(1) parser [14] into a G & G-**ifsa**, and we produce an instance of a G & G-**ifsa** which describes connectivity of speech-segments. As the result, we obtain a full PMM for the LR(1)-syntax. Finally, we derive the pattern matching algorithm for the LR(1)-syntax by applying the procedure B & B to the resultant full PMM.

The result in this paper is a refinement of the previous work [9] where we proposed a phrase-level patern matching algorithm which processes *discrete-word* sentences. In this paper, we improve it so that we obtain a pattern matching procedure which processes *continuous-word* sentences.

This paper consists of five sections. **Section 2** presents a definition of the restricted PMM and its effectiveness to represent a continuous-word recognition. **Section 3** contains an extension of the restricted PMM for representing syntax of all types. **Section 4** contains a pattern matching algorithm which processes an LR(1)-syntax.

2. A Continuous-word Recognition by Restricted PMM

In this section, we provide a definition of the pairwise Markov model (PMM) of a restricted form. We derive a restricted PMM recurrence equation, and show that it is equivalent to the recurrence equation of Sakoe's generalized two-level DP-matching (G2DP) algorithm [1].

2.1 Restricted PMM

Restricted PMM is an **fsdp** (finite state sequential decision process) Π whose base-automaton M is composed from two **fsa**'s through a kind of Cartesian product. At first, we define a *Cartesian product* of two **fsa**'s.

Def. 1. Cartesian product of two **fsa**'s

An fsa $M=(Q, I, q_0, T, Q_F)$ is composed through a Cartesian product of two fsa's $M_1=(Q_1, I_1, q_{01}, T_1, Q_{F1})$ and $M_2=(Q_2, I_2, q_{02}, T_2, Q_{F2})$ iff the following equalities are satisfied, and is denoted by $M_1 \bigstar M_2$: $Q=Q_1\times Q_2$; $I=I_1\times I_2$; $q_0=\langle q_{01}, q_{02}\rangle$; T: $Q\times I\rightarrow Q$, $T(\langle q_1, q_2\rangle, \langle a_1, a_2\rangle)=\langle T_1(q_1, a_1), T_2(q_2, a_2)\rangle$; $Q_F=Q_{F1}\times Q_{F2}$. $\times$ denotes the Cartesian product of sets. □

We then introduce a notion of a *sub-automaton* in order to deal with a constraint condition similar to the alignment-window of the discrete word recognition [2].

Def. 2. Sub-automaton

An fsa $M_1=(Q_1, I_1, q_{01}, T_1, Q_{F1})$ is called a sub-automaton of another fsa $M_2=(Q_2, I_2, q_{02}, T_2, Q_{F2})$ iff the following condition is satisfied, and is denoted by $M_1 \lll M_2$.

Condition: $Q_1\subseteq Q_2$ & $I_1\subseteq I_2$ & $q_{01}=q_{02}$ & $T_1\subseteq T_2$ & $Q_{F1}\subseteq Q_{F2}$, where $A\subseteq B$ represents either that A is a subset of B for a pair of sets A and B, or $(\forall q\in Q_2)(\forall p\in Q_2)(\forall a\in I_2)(T_1(q, a)=p\wedge p\neq q_d\Rightarrow T_2(q, a)=p)$ for a pair of transition functions T_1 and T_2 □

We may use $\times$ and $\subseteq$ instead of $\bigstar$ and $\lll$ respectively without any confusion.

Now, we define the restricted PMM.

Def. 3. Restricted PMM

Restricted PMM is an **fsdp** II whose base-automaton M is composed from the **fsa**'s M_1 and $\boldsymbol{m}_2$, and satisfies $M\subseteq M_1\times M_2$. □

In this paper, we consider only a monotonous **sdp** (**msdp**) as the PMM. Therefore, the cost function of the PMM satisfies the following monotone-conditions: ① $\xi_1\leqq\xi_2\Rightarrow h(\xi_1, q, a)\leqq h(\xi_2, q, a)$, and ② $\xi_1<\xi_2\Rightarrow h(\xi_1, q, a)<h(\xi_1, q, a)$, for any ξ_1, $\xi_2\in$ {cost value}, $q\in Q$, and $a\in I$.

The notion of the restricted PMM is shown in Fig. 1. PMM accepts both input speech and its interpretation, and outputs likelyhood of the interpretation for the input speech and a string as a subsidery output. In this figure, PMM consists of a composite **fsa** and a cost function. The cost function is computed from the input speech and a set of reference patterns, and is updated as each state transition of the **fsa**. An input string of the **fsa**, *i.e.* the interpretation, represents a syntactically legal combinations of virtual and reference words. It specifies a matching of speech segments and reference word patterns, and its cost represents adequacy of the interpretation.

PMM starts its action after reading an input speech pattern. During reading the interpretation, each state transition of the **fsa** makes update the cost function by causing to cumulate the word-distance which is specified by the input symbol. After completing the read the interpretation, PMM outputs the likelyhood of the interpretation for the input speech.

This system is a Markov model because both an **fsa** is used and the cost function is independent from its past states. Related with the combination of the input and reference patterns, a composite **fsa** in a PMM is decomposed into a pair of two **fsa**'s. In addition, the cost function depends upon the pair of states of both **fsa**'s, as well as the initial cost. Denoting the pairwise structure, we call this system the pairwise Markov model.

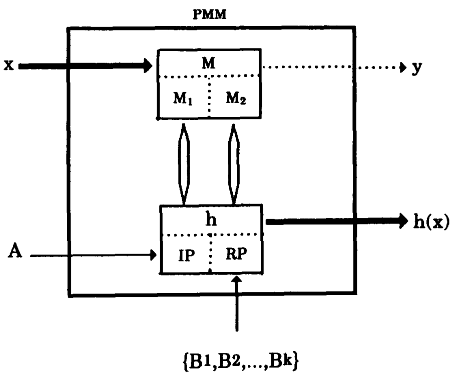

Fig. 1 Notion of the PMM where x: an interpretation; A: an input speech; Bi: a reference word pattern; M: a finite (or infinite) state automaton satisfying $M \subseteq M_1 \times M_2$; h: a cost function; y: a subsidery output string; IP: an input pattern; RP: a reference-pattern set.

2.2 PMM representation of the G2DP scheme

We show that we can derive a PMM recurrence equation comparative with the one of the G2DP algorithm. First, we review the form of the recurrence equation of the G2DP method.

We denote the input speech A as a sequence of feature vectors $a_1, a_2 \ldots a_I$, a virtual word $A(l, m)$ as $a_l a_{l+1} \ldots a_{m-1}$ which is a segment of the input speech A, a reference word pattern B^n as a sequence of another feature vectors $b_1^n b_2^n \ldots b_{J_n}^n$ which is matched to a virtual word. $D(l, m, n)$ represents a distance between a virtual word $A(l, m)$ and a reference word pattern B^n, and is computed by applying a DP-matching procedure similar to the scheme of the discrete word recognition [2].

G2DP algorithm tries to find the best interpretation for the given input speech A. For deriving the G2DP recurrence equation, the following optimization problem is dealt with [1]: find min $[\Sigma_t D(l_{t-1}, l_t, n_t) | \{l_t, n_t\}, n_1 n_2 \ldots n_k \in F(M_s)]$, where M_s is an fsa for syntax control, and $F(M_s)$ is its acceptable set, and k is a number of words contained in a sentence $n_1 n_2 \ldots n_k$. Let $G(m, q)$ be an optimum cost for the time m and the state q, and the initial condition be $G(0, 0)=0$. The recurrence equation of the G2DP method has been obtained in the following form; $G(m, q) = \min \{G(l, p) + D(l, m, n) | l, p, n | l < m, q = T_s(p, n)\}$, where T_s is a state transition function of the **fsa** M_s. □

In order to obtain a comparative PMM recurrence equation with the above one, we consider a restricted PMM defined by the following msdp Π_l: msdp $\Pi_l = (M, h, \xi_0)$, where M: an fsa (Q, I, q_0, T, Q_F), where $Q = \{\langle [1, q_1], q_2 \rangle | q_1 \in \{1, 2, \ldots, I\}, q_2 \in Q_2\} \cup \{\langle [1, 0], q_{02} \rangle\} \cup \{\langle q_{d1}, q_{d2} \rangle\}$; $I = \{\langle [i, j], a_2 \rangle | 1 \leqq i < j \leqq I, a_2 \in I_2\}$; $q_0 = \{\langle [1, 0], q_{02} \rangle\}$; $T(\langle [1, q_1], q_2 \rangle, \langle [i, j], a_2 \rangle) = \langle [1, q_1 + j], T_2(q_2, a_2) \rangle$ (for $i = q_1 + 1$); $Q_F = \{\langle [1, I], q_2 \rangle | q_2 \in Q_{F2}\}$; $M_2 = (Q_2, I_2, q_{02}, T_2, Q_{F2})$ is an **fsa** equivalent to the syntax control **fsa** M_s; h: cost function $h(\xi, q, a) = \xi + d(q, a)$; d is a non-negative function; ξ_0: an initial cost $\xi_0 = 0$. I is a number of feature vectors contained in an input speech. □

Note that I is a fixed integer because we consider merely one input speech pattern A. In this case, the following 5-tuple $M_1 = (Q_1 I_1, q_{01}, T_1, Q_{F1})$ also specifies an **fsa**, where $Q_1 = \{[1, q_1] | q_1 \in \{1, 2, \ldots, I\}\} \cup \{[1, 0]\} \cup \{q_{d1}\}$; $I_1 = \{[i, j] | 1 \leqq i \leqq j \leqq I\}$; $q_{01} = \{[1, 0]\}$; $T_1([1, q_1], [i, j]) = [1, q_1 + j]$ (for $i = q_1 + 1$); $Q_{F1} = \{[1, I]\}$. Hence, the **fsa** M is decomposed into two **fsa**'s M_1 and M_2.

The composite **fsa** M defined above accepts a finite language under the assumption that I is finite. An msdp $\Pi = (M, h, \xi)$ is called an **lmsdp** (sequential decision process of the loop-free strictly monotonous type, [11]) if its acceptable set (*i.e.*, the acceptable set of the **fsa** M) is a finite language. Hence, the restricted PMM for G2DP is an **lmsdp** and the following recurrence equation is applicable to it: let $G(q) =_{def.} \min \{h(\xi_0, q_0, x) | T(q_0, x) = q\}$ and the initial condition $G(q_0) = 0$. Hence, the PMM recurrence equation is provided for G2DP in the following form: $G(q') = \min \{G(q) + D(q, a) | q' = T(q, a)\}$. □

By substituting each elements of the **msdp** Π_l into the above PMM recurrence equation, we rewrite the above PMM recurrence equation into the following form. Let $G([1, q_1], q_2)$ be the optimum cost for both of the state q_1 and q_2, and initial cost $G([1, 0], q_{02}) = 0$. Recurrence equation is derived in the following form: $G([1, q_1'], q_2') = \min \{G([1, q_1], q_2) + D(\langle [q_1 + 1, q_1'], a_2 \rangle) | T(\langle [1, q_1], q_2 \rangle, \langle [q_1 + 1, q_1'], a_2 \rangle) = \langle [1, q_1'], q_2' \rangle\}$, where $q_2' = T_2(q_2, a_2)$. □

Comparing above PMM recurrence equation of the rewritten form with the G2DP recurrence equation, we can easily show their equivalence. Differences are caused by merely the redundant constant 1 and parenthesis [and] in the PMM recurrence equation, and different notations of elements of both **fsa**'s. Consequently, we obtain a restricted PMM recurrence equation for representing a continuous-word recognition.

The **fsa** of the restricted PMM is composed through a Cartesian product of the **fsa**'s M_1 and M_2. In the PMM recurrence equation which represents the G2DP algorithm, M_1 specifies the connectivity of segments and M_2 specifies syntax of sentences. Therefore, also in the later part of this paper, we call M_1 a *segment automaton* and M_2 a *syntax automaton*.

In this section, we assume that the PMM accepts only one input speech pattern. But, in order to regard the PMM as an acceptor of input speeches, we have to consider all possible input speeches and all their possible interpretation as input of the PMM. A way to incorporate them with PMM's is to introduce an infinite state segment automaton where $Q_{F1} =_{def.} \{[1, I_x] | I_x \in \{\text{all possible } I\text{'s}\}\}$. Moreover, by introducing an infinite state syntax automaton, we can gain a PMM which represents any syntax. In the next section, we will extend the PMM by generalizing both M_1 and M_2 (hence, M itself) into an infinite state automata (**ifsa**'s) so that we obtain the full PMM.

3. Pattern Matching Scheme

In this section, we present a pattern matching procedure which is applicable to arbitrary syntax.

We first define the full PMM. Full PMM is an infinite state **msdp** (**imsdp**) derived from the restricted PMM by replacing its **fsa** with an infinite state automaton (**ifsa**). In order that the full PMM retains the syntactic structure of the **fsdp**, we employ the ifsa originally proposed by Ginburg and Glushkov [15] which has the same *syntactic* structure as the ordinary **fsa** (*i.e.*, ifsa $M=_{def.}(Q, I, q_0, T, Q_F)$). Consequently, by reading the composite **fsa** as the composite **ifsa** (*i.e.* both segment and syntax automata are replaced by **ifsa**'s), we can consider that Fig. 1 also represents the full PMM.

The restricted PMM recurrence equation is also extended to the full PMM recurrence equation. The procedure to solve the full PMM recurrence equation, however, is inadmissible to compute because the full PMM has an infinite set of states. Because of the inadmissibility of the full PMM recurrence equations, we abandon the use of recurrence equations and concern with a branch-and-bound method in order to derive the pattern matching procedure; we apply Ibaraki's construction method for **fsdp**'s [12] to the full PMM extensively. Ibaraki's construction method requires the cost function to satisfy the monotone-condition already mentioned as well as the inequality $h(\xi, q, a)\geqq\xi$. The **msdp** Π having the cost function of this type is called a **psmsdp** (positively and strictly monotonous **msdp** [12]). In both PMM and G2DP recurrence equations, the additive cost function h also satisfies the above inequality. In the latter part of this paper, we also apply the notion of the **psmsdp** to the **imsdp** and we deal only with the **msdp** Π or **imsdp** Π of the type of **psmsdp** as the PMM.

3.1 Full PMM

An **ifsa** for the full PMM is defined below.

Def. 4. G & G-**ifsa** [15]

G & G-**ifsa** is defined by the following 5-tuple: $M=_{def.}(Q, I, q_0, T, Q_F)$, where Q: a non-empty set of states; I: a finite alphabet; q_0: an initial state; $q_0 \in Q$; T: a state transition function, T: $Q\times I\rightarrow Q$; Q_F: a set of final states, $Q_F\subset Q$. □

Following facts are presented in the reference [15]. The G & G-ifsa $M=(I^*, I, \varepsilon, T, L)$, where $T(x, a)=xa$, accepts L and L may belong to any classes. If Q is a finite set, G & G-**ifsa** is reduced to the ordinary **fsa**. T is extended to $Q\times I^*\rightarrow Q$ similar to the ordinary **fsa**. A G & G-**ifsa** is called free iff $x\neq y$ implies $T(q_0, x)\neq T(q_0, y)$ for any $x, y \in I^*$.

The name G & G-**ifsa** is due to its proposers Ginsburg and Glushkov. However, later it may be called merely an **ifsa** for simplicity.

We define the full PMM as an imsdp $\Pi=(M, h, \xi_0)$ whose **ifsa** M is composd through the Cartesian product of two **ifsa**'s M_1 and M_2. Similar to the restricted PMM, we consider that M_1 is a segment automaton and M_2 is a syntax automaton. Since an **imsdp** Π has the same syntactic structure as an **fmsdp**, every formuli defined by using an **fmsdp** are immediately extended to the corresponding **imsdp** as well as the extended formuli has the same syntactic structure as the original ones. This fact is used for the derivation of the pattern matching procedure as its basis in the next section.

3.2 Pattern matching procedure

In this section, we provide a pattern matching procedure for an arbitrary full PMM as a branch-and-bound proceudre.

Assume that the ipsmsdp $\Pi=(M, h, \xi_0)$ (*i.e.* **imsdp** Π of the **psmsdp** type) is given, where $M=(Q, I, q_0, T, Q_F)$ is an **ifsa** composed through the Cartesian product of the two **ifsa**'s M_1 and M_2. By applying Ibaraki's construction scheme [12] to the **ipsmsdp** Π, we derive the pattern matching procedure as the following branch-and-bound procedure, and name it B & B.

Def. 5. Procedure B & B

For a given full PMM represented by the ipsmsdp $\Pi=(M, h, \xi_0)$, we specify a branch-and-bound procedure B & B=(ddp Y, D, EQ, g, s) by the followings for an arbitrary search function s:

Let $M=(Q, I, q_0, T, Q_F)$ be an **ifsa**, L be an acceptable set of M, and ddp $Y=(I, L, f)$, where $f(x)=h(\xi_0, q_0, x)$ for any $x \in L$. For any $x, y \in I^*$, the *dominance relation D* is defined by $xDy=_{def.}(x\neq y)\wedge[xD_{dp}y\vee xD_{gr}y]$, where $xD_{dp}y=_{def.}[(T(q_0, x)=T(q_0, y)\wedge(h(\xi_0, q_0, x)<h(\xi_0, q_0, y))]$, $xD_{gr}y=_{def.}[T(q_0, x)\neq q_d\wedge T(q_0, y)=q_d]$, where q_d is the dead state of the M; the *equivalence relation EQ* is defined by $xEQy=_{def.}[(T(q_0, x)=T(q_0, y))\wedge(h(\xi_0, q_0, x)=h(\xi_0, q_0, y))]$; and the *lower binding function g* is defined by $g(x)=_{def.}h(\xi_0, q_0, x)$. □

The computation procedure is explicitly described as the following.

Def. 6. The procedural form of the B & B

Let B & B=(ddp Y, D, EQ, g, s), and symbols of ddp Y, D, EQ, g and s have the same definition of those in the Def. 5 respectively.

At each instant of computation of the following procedure, z denotes the smallest cost of $x \in L$ obtained by then, A denotes a set of policies which have been generated but neigther decomposed nor terminated, N denotes the set of policies which have been generated, $N-A$ denotes a set of policies which have been generated and either decomposed or terminated.

At the termination of the procedure B & B, z holds the optimum cost and x denotes the optimum sequence of pairs of the reference and virtual words.

Procedure B & B:

step 1; $z\leftarrow\infty$, $A\leftarrow\{\varepsilon\}$, $N\leftarrow\{\varepsilon\}$.

step 2; if $A=\phi$, terminate the procedure; else $x\leftarrow s(A)$ and go to step 3.

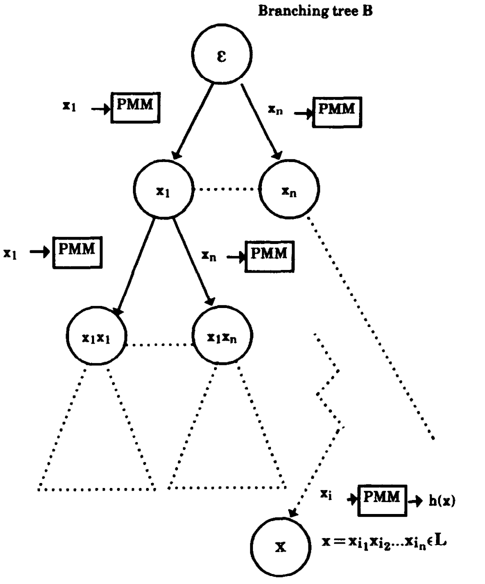

Fig. 2 Pattern matching scheme based on the PMM Input speech A for each PMM is omitted in the figure. Branching tree structure B enumerates all the possible interpretations of an input speech A. Branch-and bound procedure B & B applies the PMM paralelly through each branches of the tree B. At every node x_L, matching cost h(x) is obtained.

step 3; if $x \in L$, let $z \leftarrow \min \{z, f(x)\}$. Go to step 4.
step 4; if $g(x) > z$, go to step 8; else go to step 5.
step 5; if yDx for some $y(\neq x) \in N$, go to step 8; else go to step 6.
step 6; if $yEQx$ for some $y(\neq x) \in N - A$, go to step 8; else go to step 7.
step 7; $A \leftarrow A \cup \{xa \mid a \in I\} - \{x\}$, $N \leftarrow N \cup \{xa \mid a \in I\}$, and return to step 2.
step 8; $A \leftarrow A - \{x\}$ and return to step 2. □

Fig. 2 represents the behavior of the procedure B & B. By specifying a set of interpretations, we can construct a branching tree structure B which enumerates all the possible interpretations of an input speech A. Through applying the PMM parallelly to each nodes on the branching tree B, we can evaluate the cost function for each possible interpretation $x \in L$ and find the optimum interpretation in all legal interpretations in L.

In this section, we provide an extension of the restricted PMM, and a branch-and-bound formalization of the optimization procedure which is an alternative of the recurrence equation formalization. Note that until now the functions to compute both of the state transition function T and the judgement $q \in Q_F$ in Def. 5 and not determined yet. Hence, the procedure in Def. 6 represents a procedure rather than an algorithm, however it is applicable to any syntax. In the next section, we provide an algorithm for an arbitrary LR(1)-syntax so that we accomplish the construction of a pattern matching algorithm which processes a higher-than-the-normal syntax.

4. Pattern matching algorithm for LR(1)-syntax

Now, we present a pattern matching algorithm for LR(1)-syntax. We derive first a syntax automaton by converting an LR(1) parser [14] to an **ifsa** and then provide a segment automaton so that we obtain an PMM which accepts an LR(1)-syntax.

4.1 Syntax automaton from an LR(1) parser

We derive algorithms to compute both the state transition function T and the judgement $q \in Q_F$ by using an LR(1) parser.

LR(1) parser P_{ars} is specified by a 5-tuple (T_{bl}, D_{rv}, I_{np}, O_{ut}, W), where T_{bl} is a parsing table, D_{rv} is a driver routine, I_{np} is a input tape and contains a sequence of input symbols, O_{ut} is an output tape and contains a resultant parse-tree, W is a working area called a *stack* and contains a part of the parse-tree as intermediate information. W is modeled by a tape of semi-infinite length.

First, we define the *configuration* that is used to define a state of the syntax automation in later.

Def. 7. Configuration [14]

Let (I_{np}) be contents of the input tape I_{np}, and (W) be contents of the stack W. Configuration of an LR(1) parser is denoted by C and is defined by $C =_{def.} \langle (W), (I_{np}) \rangle$. □

Followings claims summerizes the feature of the configuration of the LR(1) parser [14].

Claim 1. At each instant of parsing, LR(1) parser computes a configuration in a finite time by looking one input symbol ahead and then the current stack-contents. The whole sequence of configurations finally obtained specifies the whole parsing process of the input sentence. □

Claim 2. At each instant of parsing, LR(1) parser produces some configurations from one input symbol. Thus, one of the following formuli is satisfied by every consecutive configuration indexed by i and $i+1$: $|(I_{np})|_i = |(I_{np})|_{i+1}$, or $|(I_{np})|_{i+1} + 1$. □

Claim 3. For the given LR(1) parser, $(W)_0$ can be taken in common for all the input sentences $x\$ \in I^*\$$. □

Claim 4. After finishing all parsing processes, LR(1) parser completes to generate the sequence of configurations $C_1C_2 \ldots C_N$ and, by using a contents of the stack in the last configuration C_N, determines in a finite time whether it accepts the input sentence $x\$ \in I^*\$$ or not. □

Let a sequence of configurations by $C_0C_1 \ldots C_i \ldots C_N$ which, generated by an LR(1) parser, discriminates a parsing process for a sentence $x\$$. This sequence is first determined, and then transformed into a sequence of states of the syntax automaton.

First, we define a transition-time when the syntax automaton transites its state. Based on the claim 2, we determine the transition-time as the time when the length of the input tape changes.

The sequence of configurations is divided into sub-sequences at each transition-time. Each sub-sequence of configurations, derived by the division, is called the ***aggregation of configurations*** and is denoted by $A_{ij;m}$. Formally, $A_{ij;m}$ is defined by the following; $A_{ij;m} =_{def.}[C_i C_{i+1} \ldots C_{i+j}]$, where $|(I_{np})|_i = |(I_{np})|_{i+k}$ for $k=1, \ldots, j$, $|(I_{np})|_i \neq |(I_{np})|_{i-1}$, and $|(I_{np})|_i \neq |(I_{np})|_{i+j+1}$ (each $|(I_{np})|_l$ contained by C_l), and m indicates an input symbol which is removed from the input tape when C_i is generated.

By the claim 1, we can state that for any input sentence $x_1x_2 \ldots x_n\$$ ($\forall x_i \in I$), the aggregation of configurations $A_{ij;m}$ is reconstructed if the top configuration C_i of the $A_{ij;m}$ is kept for every x_m, i,j. Moreover, this reconstruction requires only one input symbol x_m for every m-th step. Hence, we choose the lookahead symbol x_m instead of the input tape (I_{np}) itself and combine it with the stack content $(W)_i$ in order to obtain the representative element of the aggregation of configurations $A_{ij;m}$. Moreover, by the claim 1, we can state that the production of the sequence of configurations terminates in a finite time and that the whole parsing process of the input sentence terminates in a finite time. Hence, all the representative elements in a sequence of configurations are generated by the LR(1) parser within a finite time. Due to the above consideration, we employ the stack content $(W)_i$ in the top configuration of the $A_{ij;m}$ as the ***state descriptor*** of the LR(1) parser for an input symbol x_m in a sentence $x_1x_2 \ldots x_n\$$.

The set of states of the syntax automaton M_2 consists of all of the state descriptors. Based on the claim 3, we may take $(W)_0$ in common for all of the input sentence $x\$ \in I^*\$$. We take it as the initial state q_0 of the syntax automaton M_2. A final state of the syntax automaton M_2 is defined by a stack content $(W)_N$ contained in the last configuration C_N of the sequence $C_1C_2 \ldots C_N$. Then, the judgement $q \in Q_F$ is computable by the Claim 4.

Due to the above consideration, we construct the state transition function T for the syntax automaton M_2 by the following.

Def. 8. State transition function T

For the given LR(1) parser, let q_0 be $(W)_0$, and let q_m be $(W)_m$. The stack contents $(W)_m$ is contained by the top configuration C_i in the aggregation of configurations $A_{ij;m}$, and is generated when the m-th input symbol x_m is just input. Then, the state transition function T is defined by $T(q_{m-1}, x_m) = q_m$ for every positive integer m inductively. □

This state transition function T is computable because of the claim 1. The cardinality of the set of states computed by the above T may surely be infinite because the capacity of the stack is unlimited.

With each state descriptors, we can associate contents of the output tape generated by then. They represent a partial parse-tree (*i.e.* syntactic information) and are used for the subsidery output string of a PMM.

We must remark the following notice. If the above definition of the state transition function T implies for any LR(1) parser that $T(q, x) = T(q, y) \Rightarrow x = y$ for any $x, y \in I^*$ and $q \in Q$, the effect of the D_{dp} appearing in the def. 6 vanishes effectively. In this case, the computational efficiency of the procedure B & B is reduced to one of the simple enumeration procedure. We can prove that this case can not happen but the proof is left for another place because it requires some complicated processes.

4.2 Segment language and segment automaton

In this section, we provide an instance of a segment automaton M_1 and its special case M_1'. Since we will treat the length of input sentences as being unlimited, we use another segment language L_s different from the one that appeared in Section 2. First, we define it.

Def. 9. Segment language Ls

Let a be an arbitrary reference word, the length of the template pattern associated with it be ln (a), and σ_l, σ_u, λ_l, λ_u be finite integers respectively. The alphabet I_s for the segment language L_s is defined below: $I_s =_{def.} \{\langle \text{ln}\,(a), i, j, a\rangle | a \in I, i \in \{\sigma_l, \sigma_l+1, \ldots, \sigma_u\}, j \in \{\lambda_l, \lambda_l+1, \ldots, \lambda_u\}\}$, where I is the vocabulary for the conversation. Then, the segment language L_s is defined below: $L_s =_{def.} (I_s)^*$. □

The segment language L_s consists of sequences of symbols each of which is in a form of $\langle \text{ln}\,(a), i, j, a\rangle$; each of the elements ln (a), i, j, and a is regarded as a mere sub-symbol.

By the following reasons, I_s is a finite set. Hence, the segment language L_s is a normal language. Since ln (a) is a length of the template pattern associated with the reference word a, it is a finite integer and is determined when the pattern is registered. The integer i represents a measurement of overlapping segments which is defined by a juncture of virtual words each of which is located on the input speech consecutively. Integers σ_l, σ_u are finite constants to limit to lower and upper bound of the overlapping patterns respectively, and their values are determined empirically based on the preliminary study of speeches to be recognized. The integer j represents a measurement of the fluctuation in the length of template patterns which compensates the fluctuations in the length of input speeches. Integers λ_l, λ_u represent a lower and upper bound of such the measurement respectively and are determined by an algorithm which is inherent to the method to match the reference and the virtual word. As is shown above, parameter σ_l, σ_u, λ_l and λ_u can be determined when a registration of all the reference words has finished and are of finite values. Therefore, the vocabulary I_s is determined as a finite set before recognition computation starts.

The above segment language L_s is accepted by the following **ifsa** M_1, and it defines the segment automaton.

Def. 10. Segment automaton M_1

The segment automaton M_1 is defined as the following **ifsa** which accepts the language L_s: ifsa $M1 = (Q, I_1, q_{01}, T_1, Q_{F1})$, where $Q_1 = \{[1, 0]\} \cup \{[1, q] | q \in K\}$ (K is a set of positive integers); $I_1=\{\langle \ln(a), i, j, a\rangle \mid \ln(a)$ is an integer constant. i and j are integer variables.$\}$; $q_{01}=[1, 0]$; T_1: $Q_1 \times I_1 \rightarrow Q_1$, $T_1([1, q], \langle \ln(a), i, j, a\rangle)=[1, q+(\ln(a)+i+j)]$; $Q_{F1}=\{[1, I_x] \mid I_x \in \{\text{all possible } I's\}\}$, where I is the number of feature vectors contained in the input speech A. □

Although L_s is a normal language, we use an **ifsa** rather than an **fsa** as its acceptor because we consider all possible input speeches (note that $Q_{F1}=\{[1, I_x] \mid I_x \in \{\text{all possible } I's\}\}$). We assume that every input speech is of infinite length, and in order to extend the input speech virtually, we consider than a particular feature vector denoted by a_{nil} is added for infinite times to the actual input speech. This vector a_{nil} matches only to a particular reference word pattern \$ which represents an end-marker of a sentence and, for any other reference words, takes an infinite value of a distance function. Hence, the procedure B & B processes only an actual input speech $A=a_1a_2 \ldots a_I$ during the optimization computation if it uses a best-bound search function.

4.3 PMM for given LR(1)-syntax

During the above discussions, we have determined a pair of **ifsa**'s M_1 and M_2. Now, we can determine a base-automaton $M \subseteq M_1 \times M_2$ so as to obtain a PMM for a given LR(1)-parser. We here provide an example of the base-automaton M and then its variation.

We derive a syntax automaton M_2 by applying Def. 8 to the given LR(1)-parser P_{ars}. We then obtain the segment automaton M_1 by using the **ifsa** defined by Def. 10. Consequently, a base-automaton M of the PMM is obtained as the following form:

Def. 11. A base-automaton M for an LR(1)-syntax

Let $M_1=(Q_1, I_1, q_{01}, T_1, Q_{F1})$ be a segment automaton having a form defined by Def. 10, and $M_2=(Q_2, I_2, q_{02}, T_2, Q_{F2})$ be a syntax automaton having another form defined by Def. 8. Base-automaton $M \subseteq M_1 \times M_2$ is defined by the **ifsa** of the following form: $M=(Q, I, q_0, T, Q_F)$, where $Q=Q_1 \times Q_2$; $I=I_1 \times I_2$; $q_0=\langle q_{01}, q_{02}\rangle$; $Q_F=Q_{F1} \times Q_{F2}$; $T(\langle q_1, q_2\rangle, \langle\langle \ln(a_2), i, j, a_2\rangle, a_2\rangle) =\langle T_1(q_1, \langle \ln(a_2), i, j, a_2\rangle), T_2(q_2, a_2)\rangle$. □

In the above definition, only the symbol of the form $\langle\langle \ln(a_2), i, j, a_2\rangle, a_2\rangle$ is used for the argument of the state transition function T. This is because of the definition of the $\ln(a)$, and is a realization of $T \subseteq T_1 \times T_2$ rather than $T=T_1 \times T_2$.

Though we have already studied a discrete-word recognition in the previous paper [9], it can be considered as a special case of the above matching scheme. Discrete-word sentence is a spoken sentence and, for every words contained in such the sentence, its beginning and ending points are assumed to be already observed. In other words, it means that virtual words composing the input speech pattern are uniquely determined before starting the recognition process. Therefore, we can assign an *index number* for each virtual word and it can be used for the state of the segment automaton.

Def. 12. Segment automaton M_1' for discrete-word sentences

Let M_1' be an **ifsa** defined below: $M_1'=(Q_1', I_1', q_{01}', T_1', Q_{F1}')$, where $Q1'=\{q \mid q \in K \cup \{0\}\}$ (K is the set of all positive integers); $I_1'=\{1\}$; $q_{01}'=0$; $q_{F1}=\{q_F \mid q_F \in \{\text{all possible number of virtual words contained in the given sentence}\}\}$; $T_1'(q, a)=q+1$ for $\forall q \in Q_1'$, $\forall a \in I_1'$. □

The **ifsa** called a matching automaton in [9] is obviously derived by using the pair of M_1' and M_2.

Finally, we present a form of the cost function and obtain an PMM by combining it with the base-automaton M.

Def. 13. Cost function h

Let a base-automaton be $M=(Q, I, q_0, T, Q_F)$, and a non negative function d. The cost function h is defined as the following form:

$h(\xi, q, a)=\xi+d(q, a)$,

$h(\xi_0, q_0, xa)=h(\xi_0, q_0, x)+d(T(q_0, x), a)$. □

This form is common in the field of speech recognition where one uses a distance function as the function d. Hence, we can verify easily that it satisfies the positive and strict monotonousness conditions of the **imsmsdp**'s.

Since **ifsa** M_1 and M_2 (as well as M_1') are composed by the set of computable functions, we have finished presenting algorithms to compute both of the state transition function T and the judgement $g \in Q_F$ for the PMM of the given LR(1)-syntax. Therefore, all of the steps in the pattern matching procedure B & B become admissible to compute for the given LR(1)-syntax. Consequently, we obtain a pattern matching algorithm which accepts an LR(1)-syntax.

We finally note that the result in this section is also a new result in the field of combinatorial optimizatin because it supplies a dynamic programming algorithm for a class of set of policies belonging to the class of the LR(1)-languages, *i.e.*, higher-than-the-normal languages.

5. Conclusion

We have presented a pairwise Markov model (PMM) for deriving a new phrase-level pattern matching algorithm and obtained an algorithm which processes an LR(1)-syntax. This algorithm is usable to describe a conceptual behavior of a continuous-word recognizer which accepts an LR(1)-syntax. PMM is an acceptor of an input speech and its interpretation, and outputs likelyhood of the interpretation for the input speech. PMM has formalized as an infinite state sequential decision process **(isdp)** whose base-automaton is composed through a Cartesian product of two infinite state automata **(ifsa's)**. The syntactic structure of the PMM is the same as a finite state sequential decision process **(fsdp)**, and a finite state PMM is reduced to an **fsdp**. Concerning the pattern matching algorithm, we have considered both finite and infinite state PMM's (called a restricted PMM and a full PMM respectively). We have shown a restricted PMM recurrence equation and its equivalence to the Sakoe's G2DP (generalized two-level DP-matching algorithm [1]) recurrence equation. By using a full PMM, we construct a new pattern matching algorithm which processes an LR(1)-syntax.

We are investigating a further improvement of the PMM. In order to approach the personal speech understanding system (personal SUS), we must combine the *semantic information* processing scheme with the PMM. We now consider that the Marcus grammar [16] suggests a method for the combination of them. Marcus grammar specifies a subset of natural English in which each sentence can be parsed deterministically similar to the LR(1)-grammar. Semantic information is used to enhance the applicability of the deterministic parsing scheme to the sentences to be parsed. If a grammar of the Marcus' type could be combined with the PMM, we would obtain an improved B & B procedure on the viewpoint of the semantic information processing. Furthermore, we would use a probabilitic PMM on the viewpoint of the pragmatic information processing (which may appear in the adaptation of the recognizer to the talker as is mentioned in [17]). In this case, we may use a probabilitic automaton within the PMM instead of the deterministic **ifsa**.

We consider that we can approach a personal SUS by using a pattern matching method. PMM and branch-and-bound method provide conceptual frameworks of the personal SUS.

Acknowledgement

The author conveys sincere thanks to Dr. Tosio Kitagawa, president of the International Institute for Advanced Study of Social Information Science, for giving him a chance for the present study and continual guidances. The author also thanks heartfully Dr. Mitsuo Komura for his eager discussions, worthfull directions of this work, continuous encouragements and supports during this study. The author also acknowledges Dr. Toru Nakagawa for his help to reorganize and rewrite this paper, and critical readings of this manuscript. The author thanks to members of the pattern recognition project for their helpful advises.

References

1. Sakoe, H. A Generalized Two-Level DP-Matching Algorithm for Continuous Speech Recognition, *The Transactions of the IECE of Japan*, **E 65**, (1982).
2. Sakoe, H. and Chiba, S. Dynamic programming algorithm optimization for spoken word recognition, *IEEE Trans. Acoust., Speech, Signal Processing*, **ASSP-26-1** (1978).
3. Lowerre, B. The Harpy Speech Understanding System, in TRENDS IN SPEECH RECOGNITION, W. Lea (ed.), Prentice-Hall, Inc. (1980).
4. Jelinek, F. Continuous speech recognition by statistical methods, *Proc. IEEE*, **64** (1976), 532-556.
5. Erman, L. D. and Lesser, V. R. The Hearsay-II Speech Understanding System: A Tutorial, in TRENDS IN SPEECH RECOGNITION, W. Lea (ed.), Prentice-Hall, Inc. (1980).
6. Walker, D. E. SRI Research on Speech Understanding, in TRENDS IN SPEECH RECOGNITION, W. Lea (ed.), Prentice-Hall, Inc. (1980).
7. Wolf, J. J. and Wood, W. A. The HWIM Speech Understanding System, ibid. (1980).
8. Proceedings of ICASSP (1986).
9. Kimura, T. Branch and Bound Algorithm for Phrase Level Pattern Matching By Using Deterministic Context-Free Grammar, Research Report **60**, IIAS-SIS (1986).
10. Karp, R. M. and Held, M. Finite-state processes and dynamic programming, *SIAM J. Applied Mathematics*, **15** (1967), 693-718.
11. Ibaraki, T. Solvable Classes of Discrete Dynamic Programming, *J. Mathematical Analysis and Applications*, **43** (1973), 642-693.
12. Ibaraki, T. Branch-and-bound procedures and state space representation of combinatorial optimization problems, *Information and Control*, **36** (1978), 1-27.
13. Hopcroft, J. E. and Ullman, J. D. Formal Languages and Their Relation to Automata, Addison-Wesley Publishing Company, Inc. (1966).
14. Aho, A. V. and Ullman, J. D. The Theory of Parsing *Translation and Compiling*, **1**: Parsing, Prentice-Hall Inc., Englewood Cliffs (1972).
15. Salomaa, A. Theory of Automata, *Pergamon Press Ltd.*, (1969), *Chapter* **4**, *Section* 8.
16. Marcus, M. P. A Theory of Syntactic Recognition for Natural Language, *The MIT Press* (1980).
17. Cohen, P. S. and Mercer, R. L. The Phonological Component of an Automatic Speech-Recognition System in *Speech Recognition*, Reddy, R. (ed.), Academic Press Inc. (1975).

(Received January 24, 1986; revised June 10, 1987)