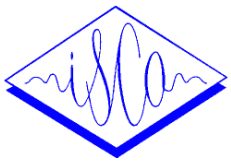
ISCA Archive
http://www.isca-speech.org/archive



3rd International Conference on
Spoken Language Processing (ICSLP 94)
Yokohama, Japan
September 18-22, 1994

Probabilistic Constraint for Integrated Speech and Language Processing

Katashi Nagao
Sony Computer Science Lab. Inc.
3–14–13 Higashi-gotanda,
Shinagawa–ku, Tokyo 141, Japan
nagao@csl.sony.co.jp

Kôiti Hasida
Electrotechnical Laboratory
1–1–4 Umezono,
Tsukuba 305, Japan
hasida@etl.go.jp

Takashi Miyata
University of Tokyo
7–3–1 Hongo, Bunkyo–ku,
Tokyo 113, Japan
mya-u@is.s.u-tokyo.ac.jp

Abstract

A totally constraint-based computational architecture is applied to integration of speech and natural language processing. A major research issue in designing information processing systems based on constraint (level of description abstracting away from information flow) is how to guarantee global adequacy of computation. A probabilistic semantics of Horn clause programs is introduced, which is a generalization of hidden Markov models, stochastic context-free grammars, etc., and a computational method for maximum likelihood estimation is proposed. This computation deals efficiently with probabilistically dependent events, and is regarded as a sort of A* search in a general sense. Furthermore, this computational architecture supports omnidirectional information flow among heterogeneous knowledge sources, from acoustics to pragmatics, and naturally resolves problems in spoken language understanding without any domain/task dependent prescription of information flow.

1 Introduction

A totally constraint-based computational architecture is employed to deal with problems in integration of speech and natural language processing. We consider a first-order, clausal-form logic program. This program is treated entirely as a constraint, in the sense that there is no predetermined direction of information flow such as top-down or left-to-right. This means that there is no domain/task-dependent procedures. The semantics of constraints is defined by a probability distribution over candidates for solutions. Symbolic computation is to transform this program so as to figure out high-probability solutions while discarding wrong hypotheses. Structure sharing in the program guarantees efficient computation for both symbolic and probabilistic information. Information processing emerges from symbolic inference driven by probabilistic computation. It is theoretically possible for our computational architecture to incorporate traditional methods of speech recognition based on hidden Markov models (HMM) (Xuang *et al.*, 1990), syntactic analysis based on stochastic context-free grammars (SCFG) (Fujisaki *et al.*, 1991), and semantic/pragmatic inference based on weighted abduction (Hobbs *et al.*, 1993). For instance, HMM consists of a finite state automaton and stochastic information, which are encoded straightforwardly in a logic program and probabilities, respectively. Therefore, our architecture is a natural extension of stochastic language models, so that it provides definite clause logic programs with similar probabilistic semantics.

A major advantage of our method over the previous ones is that it implements more fine-grained and diverse feedback loops integrating heterogeneous sorts of information, rather than constructing phoneme-level hypotheses in advance and then going on to the linguistic level as in most of the other approaches. Our system can therefore for instance pick up an alleged phoneme depending upon linguistic or extralinguistic context, even if that phoneme looks so implausible on the acoustic ground alone that traditional bottom-up architecture would filter it out before entering the linguistic mode. Note that such diverse information flow does not cause computational intractability, because information processsing at each context is centered around dynamically determined focal points or the focus of attention in the constraint network. This control mechanism gives rise to spoken language understanding without any specific procedures for specific tasks, such as matching between phonemes, parsing, and semantic and pragmatic inferences.

2 Probabilistic Constraint

A constraint is represented by a Horn clause program as the following.

(a) −p(A) −q(A).
(b) p(X) −X=f(a).
(c) p(Y) −Y=f(b).
(d) q(Z) −Z=f(U) −r(U).
(e) q(W) −W=g(V) −r(V).
(f) r(a).
(g) r(b).

A clause is basically a disjunction of *literals*. A *literal* is an *atomic constraint* preceded by a sign. An atomic constraint is an *atomic formula* such as p(X,Y,Z) or a *binding* such as X=f(Y). A sign is '−' or nil. For any atomic constraint α, literal $-\alpha$ stands for $\neg\alpha$. Names beginning with capital letters represent variables, and the other names are predicates and functors. A constraint is regarded as a network. For instance, the above constraint may be graphically represented as in Figure 1. In such a graphical representation, a clause is a closed domain containing the atomic constraints constituting that clause. Atomic constraints without such indication are referred to as negative literals in clauses. An argument of an atomic constraint is depicted as a '•'. A term like X appearing as arguments of atomic constraints in a clause is regarded as a link whose endpoints are those arguments. A link joining arguments of different clauses is called a *unification link* which represents the unifiability of terms or literals. We assume two literals are unifiable if and only if their corresponding arguments are directly connected through a unification link, and that every unification

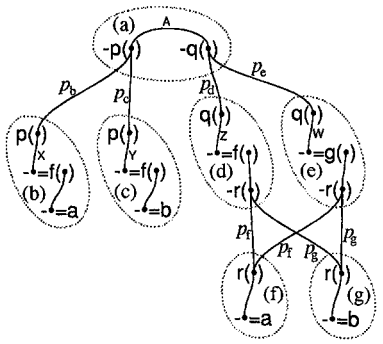


Figure 1: A graphical representation of constraint

link connects two corresponding arguments of two unifiable literals.

A *hypothesis* is a conjunction of atomic formulas and bindings. The premise of a clause is a hypothesis. A clause without positive literals is called a *top clause*. The premise of a top clause is the top-level hypothesis to be solved. In the above example, the negation of (a) $\exists A\,\{p(A) \wedge q(A)\}$ is the top-level hypothesis. We consider the minimal Herbrand models as usual. In the above example, the set of answer substitutions for p(•) is {f(a), f(b)}, and that for q(•) is {f(a), f(b), g(a), g(b)}. So, the set of answer substitutions for the whole program is {f(a), f(b)}. An *explanation* for a hypothesis is a way of combining (instances of) clauses by resolutions so as to translate the hypothesis to another hypothesis involving bindings only. We will refer to an explanation by the sequence of clauses in the order of leftmost application of resolution using their instances.[1] For example, the explanation using ⟨b, d, f⟩ corresponds to the answer f(a), while the explanation using ⟨b, d, g⟩ translates the top-level hypothesis to a wrong hypothesis. An explanation of a clause is an explanation of its premise. We will simply say 'an explanation' to mean an explanation of the top-level hypothesis. A program represents a set of explanations, and the computation as discussed later is to transform it so as to discard the explanations entailing wrong hypotheses while preserving the correct ones.

As shown in Figure 1, each unification link between each positive literal in clause Φ and other literals is assigned a *basic probability* p_Φ. The probability of each explanation is the product of the basic probabilities (of the unification links) for the clauses used therein. Thus, the probability of the explanation ⟨b, d, f⟩ is $p_b p_d p_f$.

Normalizing the sum of basic probabilities for definition clauses of each predicate less than 1 makes the sum of probabilities of solutions of each hypothesis converge to a finite number. We can therefore define the semantics of programs as a probability distribution over candidates for solutions.[2]

Finding the solution with maximum likelihood is done by checking if the explanation with the highest probability (likelihood) includes contradiction and deleting the part of program that is included only in wrong explanations. The contradiction-free explanation that found at first is one of the solutions with maximum likelihood. This computation is regarded as a sort of A* search in a general sense. In this computation, we emply a method for simultaneously evaluating as many explanations as possible by structure sharing among different explanations. One of our technical contributions is the method of efficient computation for the explanation with the highest probability (likelihood) in such structure-shared form.

[1] Here we mention the order among the literals in a clause just for explanatory convenience. This order is not significant in the computation discussed later.

[2] Such probabilistic interpretation of programs is required only when we perform a parameter learning of basic probabilities, and it is not needed in the both symbolic and analog computation discussed later.

3 Symbolic Computation

Symbolic computation begins with detection of *dependencies* between terms (i.e., variables). Dependencies trigger two sorts of symbolic operations: *subsumption* and *deletion*. Subsumption operations establish dependencies among expansions and hence probabilistic dependencies among atomic formulas.

We say that there is a *dependency between two terms* when those terms are unified in some expansion, and the sequence of terms (including them) mediating this unification is called the *dependency path* of this dependency. In Figure 1, for instance, the dependency between X and W is mediated by dependency path X·A·W. The purpose of the symbolic computation is to delete every inconsistent explanation. *Inconsistency* in an explanation occurs if two bound terms that conflict with each other (e.g., the first arguments of •=f(•) and •=g(•)) are connected through a dependency path in the explanation. Such dependency path is called an *inconsistent dependency path*. For example, in Figure 1, X·A·W is an inconsistent dependency path. The unifiability between two atomic formulas is *deleted* if, for some pair of their corresponding arguments, every dependency path involving that pair is inconsistent. Also we say a dependency path is *consistent* when all the bound terms in it are bound by unifiable bindings. Note that a solution of the program is an explanation in which every dependency path is consistent.

We consider *subsumption relation*, instead of unification, among terms. For terms ξ and η, $\xi \sqsubseteq \eta$ reads 'ξ is subsumed by η,' which means that ξ is included in η, each term being regarded as the set of its instances. Subsumption relation is useful in treating structure sharing among expansions; if we characterized the relationships among terms just in terms of unification, all the expansions must be totally separate from each other.

A *subsumption operation* creates a subsumption relationship. It is characterized by two terms: the *origin* ξ and the *target* η. The subsumption operation makes, if necessary, a copy of the clause containing η, creating therein an instance η' of η such that $\eta' \sqsubseteq \xi$.

A consistent dependency path Π motivates subsumption operations with bound terms in Π as the origins. Suppose for instance that Z is chosen as the origin for dependency path X·A·Z in Figure 1, then this path is instantiated and entirely subsumed by Z. Note that such subsumptions will eventually excavate a solution, which is an expansion involving consistent dependency paths only.

4 Maximum Likelihood Estimation

Probabilities of each explanation can be calculated as the product of basic probabilities in the explanation. Note that clause Φ may be included in several explanations. Let the **likelihood** of Φ be the probability of the explanation with the greatest probability of all these explanations. The **internal likelihood** $\mathrm{IL}(l_i)$ and the **external likelihood** $\mathrm{EL}(l_i)$ of each literal l_i in clause Φ are introduced so that the like-

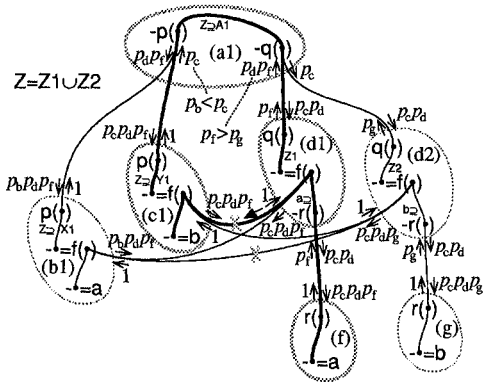


Figure 2: Immediate result of subsumption operations from Z and a

lihood of each clause can be maintained in an incremental manner when the subsumption operation changes the constraint network. Maintaining the following equations on each Φ guarantees that $\prod_{l\in\Phi} \mathrm{EL}(l)$ equals the likelihood of clause Φ, where l'' runs through literals unifiable to l.

$$\begin{aligned} \mathrm{IL}(l) &= \prod_{l'\neq l} \mathrm{EL}(l') \\ \mathrm{EL}(l) &= \max_{l''}\{\mathrm{CL}(l,l'')\} \end{aligned}$$

$\mathrm{CL}(\alpha,\beta)$ is the **connection likelihood** from α to β. When literals α and β have different signs, $\mathrm{CL}(\alpha,\beta) = p_\Phi \mathrm{IL}(\beta)$. Otherwise (i.e. α and β are bindings), $\mathrm{CL}(\alpha,\beta) = \mathrm{IL}(\alpha)/\mathrm{IL}(\gamma)$, where

- When the first argument of binding α is a member of the origin δ of the first argument of β, γ is ξ,
- The first argument of β is the member of the origin δ of the first argument of α, γ is η,

where ξ and η are bindings connected with a unification link such that the first argument of ξ is a member of δ, the first argument of η is subsumed b δ, and $\mathrm{IL}(\xi)\mathrm{IL}(\eta)$ is the greatest for all such combinations.

Let us briefly show hot to calculate each internal and external likelihood from the state after two subsumption operations at Figure 1.

In this case, the internal and external likelihoods of -p(A1) and -q(A1) in the top clause (a1) are calculated as follows:

$$\begin{aligned} \mathrm{EL}(\mathsf{p_{a1}}) &= p_\mathrm{c} \max\{\mathrm{IL}(\mathsf{p_{b1}}), \mathrm{IL}(\mathsf{p_{c1}})\} = p_\mathrm{c} \\ \mathrm{EL}(\mathsf{q_{a1}}) &= p_\mathrm{d} \max\{\mathrm{IL}(\mathsf{q_{d1}}), \mathrm{IL}(\mathsf{q_{d2}})\} = p_\mathrm{d} p_\mathrm{f} \end{aligned}$$

So, the likelihood of the top clause (a1) is $\mathrm{EL}(\mathsf{p_{a1}}) \times \mathrm{EL}(\mathsf{q_{a1}}) = p_\mathrm{c} p_\mathrm{d} p_\mathrm{f}$. This likelihood is the probability of a wrong explanation, because the unification link between binding •=f(•) in clause (c1) and binding •=f(•) in clause (d1) contradicts. There is another contradiction of the unification link between binding •=f(•) in clause (b1) and binding •=f(•) in clause (d2). When these two contradicting unification links are deleted, the internal and external likelihoods are updated as shown in Figure 3.

Now let us suppose $p_\mathrm{b} p_\mathrm{f} > p_\mathrm{c} p_\mathrm{g}$. Then, The external likelihoods of the literals in the top clause (a1) are as follow:

$$\begin{aligned} \mathrm{EL}(\mathsf{p_{a1}}) &= \max\{\mathrm{CL}(\mathsf{p_{a1}},\mathsf{p_{b1}}), \mathrm{CL}(\mathsf{p_{a1}},\mathsf{p_{c1}})\} = p_\mathrm{b} \\ \mathrm{EL}(\mathsf{q_{a1}}) &= \max\{\mathrm{CL}(\mathsf{q_{a1}},\mathsf{q_{d1}}), \mathrm{CL}(\mathsf{q_{a1}},\mathsf{q_{d2}})\} = p_\mathrm{d} p_\mathrm{f} \end{aligned}$$

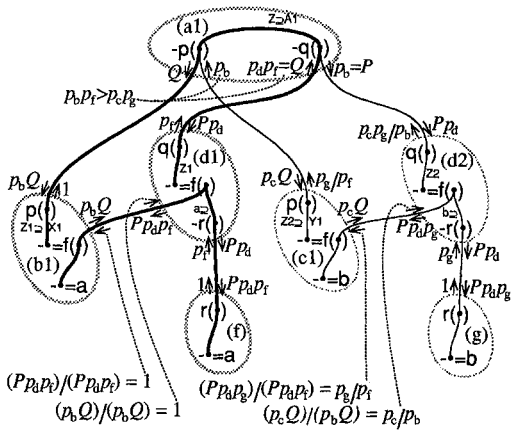


Figure 3: After subsumption operations from Z and a and deletion

In Figure 2, there exists a whole explanation that includes any combinations of the explanations of $\mathsf{p_{a1}}$ and $\mathsf{q_{a1}}$ so that $\mathsf{p_{a1}}$ and $\mathsf{q_{a1}}$ are probabilistically independent. On the contrary, Figure 3 entails only two explanations (⟨b1, d1, f⟩ and ⟨c1, d2, g⟩) so that $\mathsf{p_{a1}}$ and $\mathsf{q_{a1}}$ are probabilistically dependent. Although there arise no such probabilistic dependencies in the cases of HMM and SCFG, probabilistic Horn constraint appears to require such a treatment of probabilistic dependencies for the sake of efficiency.

5 Linguistic Constraints

Natural language processing (NLP) has employed statistical tools such as HMM, SCFG, stochastic tree-adjoining grammars (STAG) (Shabes, 1992), probabilistic generalized LR parser (PGLRP) (Wright and Wrigley, 1991), etc. **Probabilistic Constraint** subsumes all of them.

Input-bound probabilistic logic programs can encode various constraints needed for NLP, including HMM, SCFG, STAG, and PGLRP. We say that a program is *input-bound* if every wrong expansion contains a conflict involving a binding in the top clause.

To begin with, the following clauses encode a part of an HMM, which at state s goes to state t with probability p_a and to state u with probability p_b, and outputs a with probability p_c and b with probability p_d.

(a) s(X) -s_out(X,Y) -t(Y).
(b) s(X) -s_out(X,Y) -u(Y).
(c) s_out(X,Y) -X=a(Y).
(d) s_out(X,Y) -X=b(Y).

There is a standard way of encoding a CFG in an input-bound program (Pereira and Warren, 1980). In fact, the following input-bound program encodes the problem of syntactic parsing of string *aba* under the context-free grammar (CFG) consisting of production rules such as $S \to a$, $S \to ST$, and so on.

(a) -s(A0,z) -A0=a(A1) -A1=b(A2) -A2=a(z).
(b) s(B,C) -B=a(C).
(c) s(D,E) -s(D,F) -t(F,E).
(d) s(G,H) -u(G,I) -s(I,H).
(e) t(J,K) -J=b(L) -s(L,K).
(f) u(M,N) -s(M,O) -O=b(N).

6 Integrating Semantic and Pragmatic Information

We consider the semantic/pragmatic processes along with an example in which the following Japanese sentence is understood from an acoustic input.

tegami kaita
letter wrote
'(I) wrote a/the letter.'

Since phoneme /k/ and /h/ are similar to each other, *kaita* might sound like *haita* (vomited). In addition, *kaita* might also sound like *kaite* (write-IMPERATIVE) due to the similarity between /a/ and /e/. The former confusion is resolved by selectional restrictions, and the latter by the semantic and pragmatic context.

The constraint for semantic processing is as follows:

(1) –sentence(S,X,_) –speech0(X,Y) –sounds(Y).

(2) np(S,X0,X6) –letter(S) –t(X0,X1) –e(X1,X2) –g(X2,X3) –a(X3,X4) –m(X4,X5) –i(X5,X6).

(3) vp(S,X0,X5) –write(S) –past(S) –k(X0,X1) –a(X1,X2) –i(X2,X3) –t(X3,X4) –a(X4,X5).

(4) sentence(Event,X,Z) –np(Index,X,Y) –vp(Event,Y,Z) –sbjobj(Event,Index).

(5) sbjobj(Event,Index) –agent(Event,Index).

(6) agent(_,X) –animate(X).

(7) sbjobj(Event,Index) –patient(Event,Index).

(8) patient(Event,Index) –write(Event) –document(Index).

Now, we explain a semantic process using clauses (5, 6, 7, 8). (5) means that if Index is Event's **agent**, then Index is Event's **subject** or **object**. Similarly, (7) means that if Index is Event's **patient**, then Index is Event's **subject** or **object**. For semantic processing, we introduce a semantic consistency that is a slight extension of the unifiability. Two terms that are connected via a dependency path are *semantically consistent* if one is subordinate in thesaural relations to the other. Due to the subsumption operation, to make sbjobj(V,N) true in (4) requires that Index be an **agent** or **patient** of Event. From (6), if Index is **agent**, then Index should be **animate**. However, **letter** is not semantically consistent with **animate**. Thus, dependency path Y·D·A·Z in Figure 4 causes a contradiction. Then, dependency path Y·D·A·Z becomes less activated than dependency path X·C·A·Z, which does not contain contradictions. In general, an interpretation with more contradiction-free dependency paths is preferable. This heuristics emerges from the probabilities of constraints. In this example, the interpretation *tegami kaita* ('(I) wrote a/the letter') is the most preferable, because it involves more contradiction-free dependency paths than other interpretations such as *tegami haita* ('(I) vomited a/the letter'), which has no dependency path, and *tegami kaite* ('please write a letter'), which involves two dependency paths (which are the same as the interpretation *tegami kaita* except dependency path W·B·E).

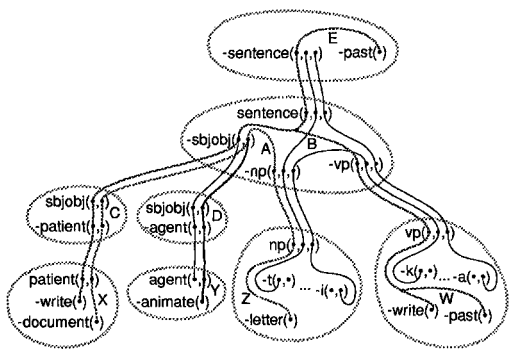


Figure 4: Interaction among syntax, semantics, and pragmatics

If we have some pragmatic information indicating that the speaker is requesting something, then the first interpretation will have only two cycles while the third interpretation will have a higher probability (likelihood). Thus, the third interpretation (i.e., the imperative interpretation) will be preferred. The semantic/pragmatic constraints can, thus, contribute to prediction/activation of the interpretation of utterances.

7 Concluding Remarks

The proposed method provides a general framework to integrate diverse types of computation. The homogeneous representation enables global application of a general control scheme, such as A* search, simulated annealing, and so forth. A key technical issue there is the efficient treatment of probabilistically dependent literals.

We are planning to implement the present theory on a massively parallel machine and include the acoustic computation into the constraint-based architecture, so as to directly incorporate it into the tight feedback loops realized by the constraints. This will necessitate a learning algorithm to acquire the probabilistic properties of the constraint on the linguistic sounds. Our conjecture is that such parameter learning is possible by using the generalized backpropagation method (Pineda, 1988).

References

T. Fujisaki, F. Jelinek, J. Cocke, E. Black, and T. Nishino. 1991. A probabilistic parsing method for sentence disambiguation. In Masaru Tomita, editor, *Current Issues of Parsing Technology*, pages 139–152. Kluwer Academic Publishers.

J. R. Hobbs, M. E. Stickel, D. E. Appelt, and P. Martin. 1993. Interpretation as abduction. *Artificial Intelligence*, 63(1–2):69–142.

F. C. N. Pereira and D. H. D. Warren. 1980. Definite clause grammars for language analysis — a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278.

F. J. Pineda. 1988. Generalization of backpropagation to recurrent and higher order neural networks. In D. Z. Anderson, editor, *Neural Information Processing Systems*, pages 602–611.

Y. Shabes. 1992. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the Fourteenth International Conference on Computational Linguistics (COLING-92)*, pages 426–432. Association for Computational Linguistics.

J. H. Wright and E. N. Wrigley. 1991. GLR parsing with probability. In Masaru Tomita, editor, *Generalized LR Parsing*. Kluwer Academic Publishers.

X. Xuang, Y. Ariki, and M. A. Jack. 1990. *Hidden Markov Models for Speech Recognition*. Edinburgh University Press.