TR-0740

論理プログラムの並列帰納学習システムの構築

坂本　忠昭（三菱）

February, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

論理プログラムの並列帰納学習システムの構築

An Implementation of Parallel Inductive Learning System for Logic Programming

坂本 忠昭
Tadaaki Sakamoto
三菱電機 (株) 中央研究所
Mitsubishi Electric Corporation

Abstract: An inductive learning system for logic programming is a system which induces logic program from given examples, e.g. a set of ground atoms, and many systems have been proposed recently. Several problems, however, have been pointed out. One of most significant problems is that systems rely heavily on teachers to answer to any difficult questions or to provide complete background knowledge to guide induction. And another problem is that a heuristic search may sacrifice a possibility of finding better hypotheses than first one. This paper presents how to parallelize inductive learning algorithm to decrease a load of teacher and to search all hypotheses that satisfy all given examples. And the efficiency of the parallelized system is shown through some examples.

1 はじめに

帰納学習とは一般に、与えられたデータ（例）集合からそれを説明する一般則を生成する学習である。あるいは、背景知識 K 及び $K \not\vdash E^+$ である正の例の集合 E^+ と負の例の集合 E^- を与え、$K \cup H \vdash E^+$ かつ $K \cup H \not\vdash E^-$ を満たす仮説知識 H を生成することであると言うこともできる。そして、論理プログラムの帰納学習の場合には、正の例及び負の例は一般に基底アトムであり、背景知識及び仮説知識は節集合（論理プログラム）となる。論理プログラムの帰納学習は、帰納学習の対象を論理プログラムに限定することにより、一階述語論理の性質を活かした効率的な帰納推論方式の確立を目指すものである。

論理プログラムの帰納学習に関しては、Shapiro の MIS[11] や Muggleton と Buntine の CIGOL[6] を初めとする数多くのシステムが発表されてきた [7][8][9]。しかし、目的の論理プログラムを本当に効率良く生成するための手法はまだ確立されていない。多くのシステムでは効率を得るために、教師に対し探索制御に直接関わるような質問を行なったり、十分な背景知識を準備させることによって探索空間を狭めたり、ヒューリスティクスを用いた探索制御を行なったりしている。ところが、これらの方法では、質問に答えたり背景知識を準備したりする教師の負担が大きすぎるという点や、ヒューリスティクスによる枝刈りによって、より良い仮説知識の生成される可能性が失われてしまうという点が指摘されている [4][9][5]。

これらの問題に対して筆者は、並列処理を用いることによって、教師にかかる負担を軽減し、かつ有効な仮説知識を効率的に全て求めることを目指す並列帰納学習システムの構築を行なってきた [10]。本稿では、この並列帰納学習システムの概要を述べると共に、並列推論マシン上への実装に関して、特に探索の効率化について述べる。

2　システム概要

2.1　並列化について

まず最初に、なぜ帰納学習システムを並列化をするかという点について簡単に述べる。従来システムの問題点であった教師の負担を軽減するということは、教師が従来行なっていた探索制御を減らすあるいは放棄するということに等しい。従って、これは探索空間の拡大や探索木の分岐点における非決定性の発生の原因となる。さらに、これは別解の探索を行なおうとする場合も同様に発生する問題である。つまり、従来は教師やヒューリスティクスのガイド付きで縦型探索（しかも仮説知識が１つでも見つかれば終了という探索）をしていたものを、一気にガイド無しの全解探索にするということである。しかも、このように探索空間が広がっても、短時間で解を探索するという要求は依然として残っている。従って、このような要求を満たすためには、広くしかも非決定的な分岐を持つ探索空間を高速に探索する方法に基づいて帰納学習システムを構築する必要がある。帰納学習システムへ並列処理を導入することにしたのは、並列処理がこの要求を満たす方法であると考えられるためである。

2.2　並列帰納学習アルゴリズム

並列帰納学習の全体的な枠組を図 1に示す。背景知識をもとに帰納操作によって仮説知識を生成する処理が学習の１サイクルになる。各背景知識に対する処理はお互いに独立であるため並列に実行される。学習の各サイクルにおいて、システムは教師に対して３種類の例の入力を要求する。ただし、教師は必ずしも３種類全ての例を入力する必要はない。これらの例のうち、具体例と正の例はどちらも学習対象概念の正の例であるが、具体例はその概念を生成する際に帰納操作の適用対象となるのに対し、正の例は生成された仮説知識のチェックにのみ用いられ帰納操作の適用対象にはならない点が異なる。また、負の例も正の例と同様に仮説知識のチェックにのみ用いられる。

次に、各背景知識に帰納操作を適用する際のアルゴリズムを図 2に示す。アルゴリズムは大きく２つの部分に分けられる。前半は現在対象としている節集合が正の例や負の例を満たすかどうかチェックする部分である。負の例は仮説知識が満たしてはならない例であるから、満たしている場合には失敗終了し、それ以上の探索は行なわれない。負の例を満たさず

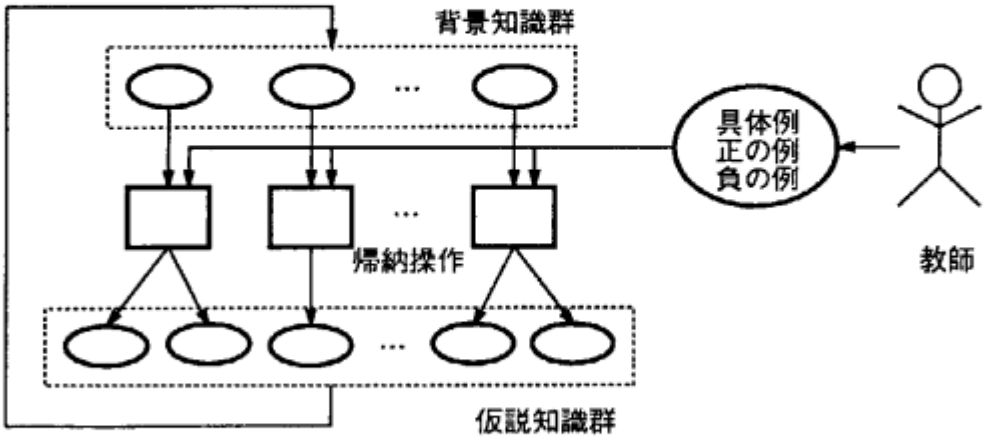

図 1: 並列帰納学習の枠組

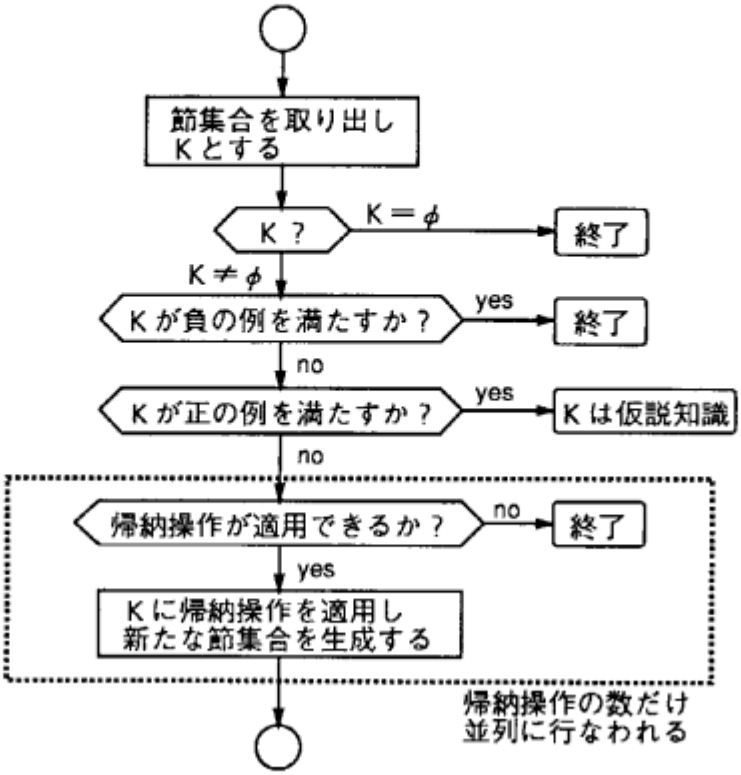

図 2: 帰納操作適用アルゴリズム

正の例を満たすものは目的とする仮説知識となる。負の例も正の例も満たさないものは、まだ帰納操作の適用が不十分であると考えられるため、後半の帰納操作適用部へと進む。後半は、前半のチェックによって選択された節集合に対して帰納操作を適用する部分であり、実際にはシステムが準備している帰納操作の数だけ並列に実行される。これは各帰納操作がお互いに独立に実行可能であるためである。本システムでは 4 種類の帰納操作を準備しているため、この部分は 4 つの処理が並列に実行される。帰納操作の適用の結果幾つかの節集合が生成されるが、各節集合に対してこのアルゴリズムを再帰的に適用していくことになる。

なお、一般の帰納学習システムでは具体例が入力されると直ちに帰納操作が適用されるのに対し、本システムは背景知識によって導くことができない正の例が存在して初めて帰納操作が適用されるという方針をとっている。また、先に述べたようにその正の例を帰納操作適用の終了条件として用いている。すなわち、正の例を満たす仮説知識が生成された時点で帰納操作の適用を終了するというものである。これは、本システムにおける帰納操作が全て一般化操作であるため、操作の過剰な適用による過度の一般化 [1] を防ぐ目的で行なわれているものである。

2.3 帰納操作

帰納操作としては、基本的には CIGOL で用いられている 3 つの操作 truncation, absorption, intra-construction を用いているが、オリジナルのアルゴリズムには非決定的要素が多く、そのままでは探索空間が非常に広がるため、新たに幾つかの制限を追加している。また、次節で述べる文法学習の例題用に intra-construction を特化した帰納操作を追加してい

る。以下各帰納操作について簡単に述べる。

truncation

2つの単一節間の最小一般化を行なう操作である。ただし、生成された一般化節のうち、それに含まれる変数が全て異なるようなものは削除する。これは、そのような節が一般的過ぎて意味を持たないと考えられるからである。なお、この制約は他の操作についても同様である。

absorption

2つの節 C_1, C_2 から導出によって節 C が導かれるという関係を仮定し、与えられた C_1 と C から C_2 を求める操作である（図 3参照）。ただし、C_1 は単一節に制限されている。さらに、本システムでは absorption を自己再帰節を生成する操作として限定している。これによって、C の頭部と C_1 との述語記号及びアリティが等しいことが操作適用の条件となっている。

intra-construction

単一節でない節 A と単一節集合 $B_1, B_2, \ldots, B_m$ から節集合 $C_1, C_2, \ldots, C_m$ が導出されるという関係を仮定し、与えられた $C_1, C_2, \ldots, C_m$ から A と $B_1, B_2, \ldots, B_m$ を求める操作である（図 3参照）。求められた節には、新しく生成された述語が含まれている。CIGOL では、新しい述語が生成された場合には教師に対してその述語名の問い合わせを行なっているが、本システムでは行なっていない。なお、新しく生成された述語が背景知識に既に存在すると判断した場合には、システムが自動的に述語名の付け替えを行なう。

intra-construction-g

節集合 $C_1, C_2, \ldots, C_m$ が与えられたときに、intra-construction がこれら全てに対して適用するのに対し、その一部に対して適用する操作が intra-construction-g である。操作の内容は intra-construction と同様であるが、適用対象節の組み合わせ数の増加を抑えるため、そしてこの操作が次節の文法学習問題用であるため、入力される節の形式が

$$\mathrm{sentence}([t_1, \ldots, t_h|X], X)$$

に制限されている。ただし、$t_1, \ldots, t_h$ $(h \geq 1)$ は変数を含まない項、X は変数である。

3 システムの効率化

前節で述べた並列化に基づき、並列論理型プログラミング言語 KL1[2] を用いて、システムの並列推論マシン Multi-PSI 上への実装を行なった。ここでは、並列マシンによる実

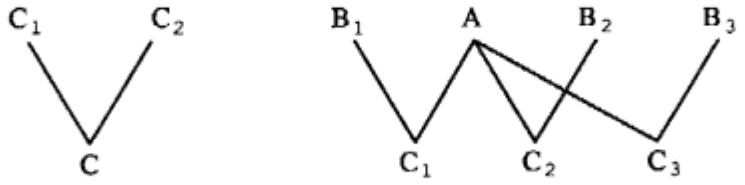

図 3: absorption と intra-construction

際の処理効率がどの程度のものであるか、また、実装時にどのようにしてその効率を向上させるかといった点について、例題を通して述べていく。

3.1 例題

例題として、簡単な文法学習問題を考える。これは、

具体例：
(1) sentence([det, noun, verb|X],X).
(2) sentence([det, noun, verb, det, noun|X],X).
(3) sentence([det, noun, verb, det, adj, noun|X],X).
(4) sentence([det, adj, noun, verb|X],X).
(5) sentence([det, adj, noun, verb, det, noun|X],X).
正の例：
(6) sentence([det, noun, verb, det, adj, adj, noun|X],X).
(7) sentence([det, adj, adj, noun, verb, det, adj, adj, noun|X],X).

といった文の例を与え、

```
sentence(X,Z) :- np(X,Y),vp(Y,Z).
np([det|X],Y) :- n(X,Y).
n([noun|X],X).
n([adj|X],Y) :- n(X,Y).
vp([verb|X],X).
vp([verb|X],Y) :- np(X,Y).
```

という文法を求めるものである。なお、簡単のため文は上記のように品詞の列として与えるものとする。この問題では、のべ671回の帰納操作の適用が行なわれ、全部で70個の仮説知識が生成される。ただし、この中には同じものが含まれており、それらをまとめると10個になる。現在は処理の最後でこの70個の仮説知識をフィルタに通して同じものを取り除いている。10個の仮説知識のうちの1つは上記の文法を表しており、その他の9つはこれとは少しずつ形の違うものとなっている。しかし、いずれも具体例及び正の例を満たしているという点では正しい仮説知識といえる。

3.2 負荷分散

並列実装の際には、負荷分散方法が非常に重要な問題となる。負荷分散は具体的には、タスクをどのようなサブタスクに分割し、それをどのプロセッサ要素（PE）に割り付けるかという問題であるということができる。ここで、実装に用いた並列推論マシン Multi-PSI（16PE版）のPE構成を図4に示す。丸がPEを表しその中の数字がPE番号を表す。PEは図のようにメッシュ結合されており、1番のPEにフロントエンドプロセッサ（FEP）がつながっている。

まずタスクの分割については、その大きさ（粒度）の違いによって2種類を考えた。粒度の大きいものは、図2で示した帰納操作適用アルゴリズム全体をサブタスクとするものであり、粒度の小さいものは、このアルゴリズム後半の各帰納操作の適用部を操作毎に別々の

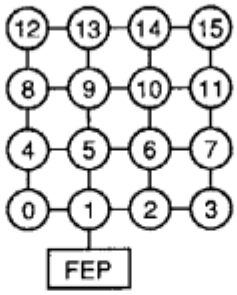

図 4: Multi-PSI の PE 構成

PE に割り付けるものである。次に PE への割り付け方法については、基本的には動的割り付けを行なうことにし、サイクリック割り付けとの比較を行なうことにした。動的割り付けは、割り付け要求が発生するとそのとき暇な PE を探してタスクを割り付けるという方式であり、OR 並列型全解探索問題に適していることが報告されている [3]。サイクリック割り付けは、割り付けの要求が発生する度に順に 1 だけ大きい番号を持つ PE に割り付けるという方式である（勿論 15 番を越えた場合には 0 番に戻る）。いずれの場合にも PE 番号を決定するプロセスは全体で 1 つであり、最初にタスクが投入される PE（通常は 0 番）に常駐している。

以上のサブタスクの粒度（小、大）と割り付け方（サイクリック、動的）の 4 つの組合せについて、PE 数を 1, 2, 4, 8, 12, 16 に変えて実行時間を計測した結果を表 1に示す。表中の括弧内の数値は 1PE の処理に対する台数効果である。また、PE 数 1 は PE 番号 0 の PE のみを使用したものであり、PE 数 2, 4, 8, 12, 16 は、それぞれ PE 番号 0 から 1, 3, 7, 11, 15 までの PE を使用したものである（図 4参照）。表より、動的割り付けがサイクリック割り付けよりも効率が良く、粒度の大きいものの方が小さいものよりも効率が良いことがわかる。なお、PE 数 1 の場合には、割り付け先の PE は常に 0 番であるが、番号決定のためのプロセスは動いている。さらに、表 1をグラフに表したものが図 5及び図 6である。

ここで、なぜ動的割り付けや粒度が大きい方の効率が良いのか考えてみる。まず、動的割り付けについて、Multi-PSI に用意されている runtime monitor による実行時のモニタ結果を図 7に示す。これは、単位時間（ここでは 2 秒）における全 PE の稼働状態を濃淡グラフで表示したものである。図の前半が動的割り付け、後半がサイクリック割り付け（どちらも粒度大で PE 数は 16）の実行結果である。動的割り付けでは全 PE に均等に負荷が

表 1: 実行時間 (msec) と台数効果

PEs	cyclic +粒度小		cyclic +粒度大		dynamic +粒度小		dynamic +粒度大	
1	76365	(1.00)	75352	(1.00)	77570	(1.00)	76150	(1.00)
2	58872	(1.30)	47787	(1.58)	58193	(1.33)	46198	(1.64)
4	36999	(2.06)	29056	(2.59)	34155	(2.27)	25096	(3.03)
8	23413	(3.26)	16367	(4.60)	18771	(4.13)	14013	(5.43)
12	17310	(4.41)	12588	(5.99)	13283	(5.84)	9550	(7.97)
16	14092	(5.42)	11020	(6.84)	10480	(7.40)	7510	(10.14)

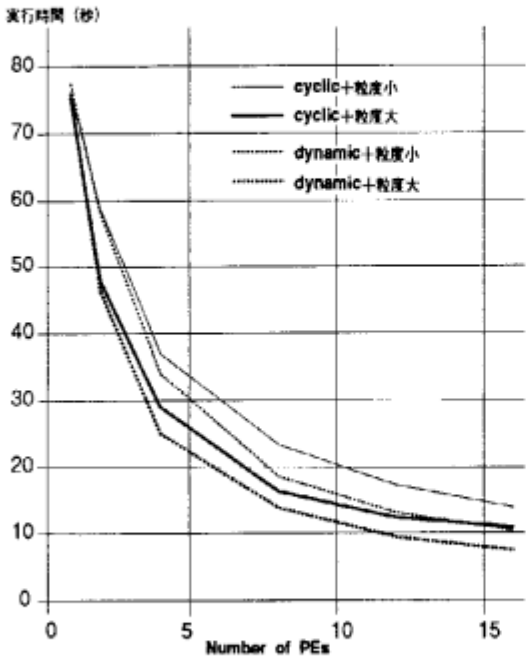

図 5: 実行時間

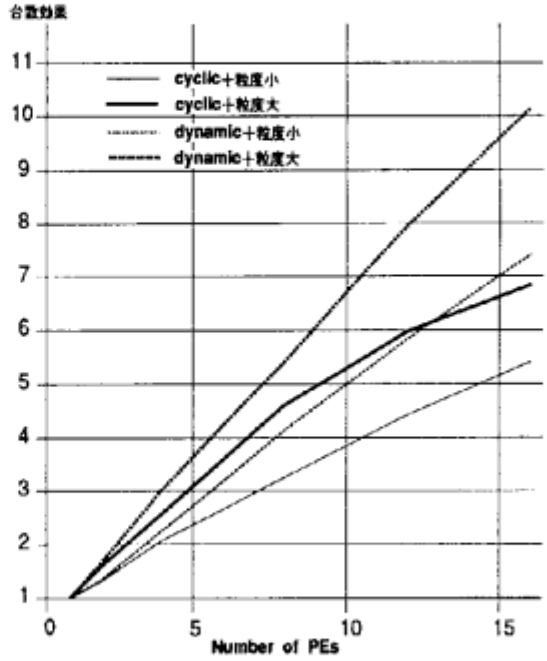

図 6: 台数効果

分散されているのに対し、サイクリック割り付けでは負荷の分散に偏りがあることがわかる。これは、サイクリック割り付けがその PE の忙しさにかかわらずタスクを割り付けてしまうためである。図 6において、サイクリック割り付けで粒度が大きいものの効率の伸びが PE 数の増加と共に著しく鈍ってくるのは、粒度が大きい場合にはこの偏りが PE 数の増加に伴って顕著に現れてくるためと考えられる。

次に粒度について、やはり Multi-PSI に用意されている runtime average monitor による実行時のモニタ結果を図 8に示す。これは、単位時間における全 PE の平均稼働率を示すものであるが、稼働状態として計算、受信、送信、GC を別々に表示することができるも

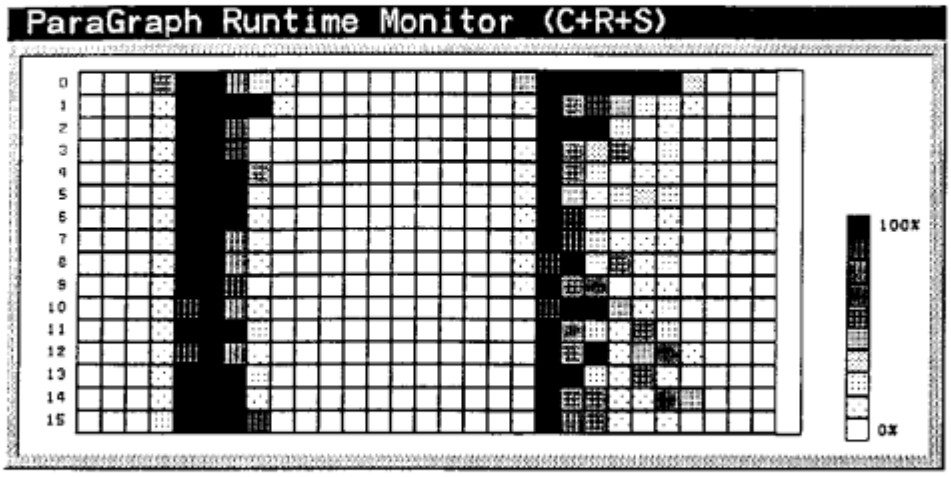

図 7: 割り付け方法の比較

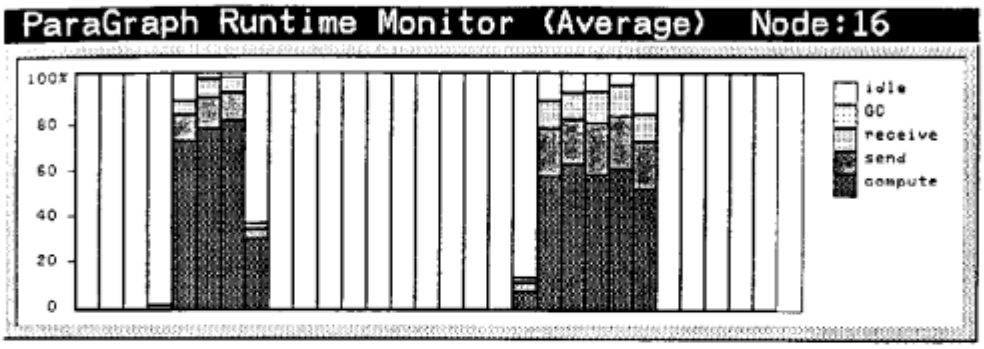

図 8: タスク粒度の比較

のである。図の前半が粒度大、後半が粒度小（どちらも動的割り付けで PE 数は 16）の結果である。図を見ると、どちらもピーク時の稼働率は 100% に近いものの、粒度が小さい方は通信の割合が大きいことがわかる。つまり、粒度が小さいものは、多くのサブタスクが生成されるため、割り付けの際のデータ通信や、動的割り付けのための暇な PE の探索などによる PE 間の通信量が多くなることによって、計算処理が圧迫され効率が低下すると考えられる。

3.3 問題分割

実装に直接関係するものではないが、同じ問題でも例の与え方によって実行効率を向上させることができるということについて述べる。先に示した例題において、まず (1) ～ (3) の具体例と (6) の正の例を入力として仮説知識を生成し、次にそれを背景知識として (4),(5) の具体例と (7) の正の例を入力して仮説知識を生成するといった 2 段階の学習を考える。このとき、例を分割しない場合と分割した場合とで探索空間の大きさ及び実行時間を測定したものを表 2に示す。

表より、問題を分割して学習させた方が探索空間も処理時間も大幅に縮小できることがわかる。これは、問題を分割することにより、システムの扱う節集合内の節の数が減少するため、帰納操作の絶対的な適用数が減ることによると思われる。つまり、探索木において各分岐点における分岐数が全体的に減少するため、探索空間の広がりが抑えられるということになる。なお、仮説知識数が 10 から 8 へ減少しているが、除かれた 2 つの仮説知識は、システムが自動的に生成する新述語名を付け替えることによって他の仮説知識と同じになるものである。本システムのフィルタは現在のところ、このようなタイプの「同じ」仮説知識を除くことができない。これを除く処理はまた 1 つの組合せ問題であるため、効率化を考慮した上で実装する必要があり、今後の課題の 1 つである。

表 2: 問題分割の効果

	探索木のノード数	仮説知識数	実行時間 (msec)
問題分割なし	671	10	7510
問題分割（1 段階目）	24	8	583
問題分割（2 段階目）	86	8	2092

4　評価及び検討

まず、本システムの目的である教師の負担の軽減と全解の探索についてであるが、教師は最初に例を入力するだけで良く、システムから何の質問も受けることはない。CIGOLでは、アーチ問題[6]のような簡単な例題でも教師に仮説の真偽を6回も問い合わせていることを考えると、教師の負担は軽減されたといえる。さらに、システムは教師に質問をせずに与えられた例を全て満たす仮説知識を生成することができた。ただし、完全性の証明がされていないので、正確には全解とは言えないかも知れない。

実装時の負荷分散に関しては、帰納操作適用アルゴリズムを1つのタスクとして、動的割り付け方法を用いた場合が最も効率が良く、16台で10倍以上の台数効果が測定された。これは、並列帰納学習が動的割り付けが効果的と言われるOR並列型全解探索問題そのものであることと、サブタスクの粒度が適度なものであったことによると思われる。さらに、図6において動的かつ粒度大の場合にはPE数の増加と共に台数効果がほぼ直線で増加していることから、PE数を16台以上に増やすことによって、この比率に従ってさらに実行時間を短縮できることが期待される。

また、絶対的な実行時間についてみても、例題で示した規模の問題（全帰納操作数671）を16台のPEを用いることによって7.5秒で仮説知識を生成することができた。他の幾つかの例題の実行結果から全帰納操作数が1600近い問題も約20秒で仮説知識を生成できることがわかった。これらの例題だけで全てを判断することは難しいものの、前節で述べた問題の分割を併用することによってかなりの規模の問題を短時間で解くことが期待される。逆に教師の立場から考えると、複雑な概念を学習させる場合には、概念を分割しサブ概念を順番に学習させるとか、まず基本部分を学習させて次第に周辺部分を学習させるといった段階的な学習方法をとることになると思われる。従って、このような多段階の学習方式が重要になると共に、多くの例が与えられた場合にはシステムが自動的に例を分類し、段階的な学習を行なうといった手法も必要になってくると思われる。

5　おわりに

論理プログラムを帰納的に学習するシステムの並列化及びその効率化について述べた。最初に述べたように、本システムは教師の負担を減らし、かつ全解を探索することを目的としているが、これらはいずれも探索空間の拡大と非決定性の増加につながる問題である。一般にはこのような問題に対しては、各帰納操作の改良によって探索空間をあまり広げずに効率良く解を求めようとしている。これに対し、本システムはこの広がった探索空間を並列処理の持つ計算能力を用いて言わば力ずくで探索し、効率を上げようとするものである。帰納学習アルゴリズム自身の持つ並列性によって、並列推論マシン上への実装の結果、処理時間や効率についてはある程度期待通りの結果が得られたと思われる。現在のところ、帰納推論に直接関係する各帰納操作については、若干の改良は施したものの本質的な部分における検討はまだ行なっていない。従って、帰納操作の改良によってさらなる効率の向上が期待できる。反面、探索空間の拡大は一般に指数関数的であるため、問題を複雑にしていくと、単純に例の数を増やしただけでもたちまち並列処理能力の限界を越えてしまうことが予想される。そのような点に対しては、例えば問題を分割して多段階の学習にもっていくといった手法が有効であると考えられる。

謝 辞

本研究は第5世代コンピュータ・プロジェクトの一環として行なわれた。日頃御指導頂いているICOTの古川康一研究所次長、長谷川隆三第5研究室室長に感謝致します。

参考文献

[1] Bain,M.: Experiments in Non-monotonic First-order Induction, *Proc. of ILP-91,* 1991, pp.195–206.

[2] Chikayama,T.,Sato,H.and Miyazaki,T.: Overview of the Parallel Inference Machine Operating System (PIMOS), *Proc. of Int. Conf. on FGCS'88,* 1988, pp.230–251.

[3] 古市昌一, 瀧和男, 市吉伸行：疎結合並列計算機上でのOR並列問題に適した動的負荷分散方式とその評価, *Proc. of KL1 Programming Workshop '90,* 1990, pp.1–9.

[4] 石坂裕毅, 有川節夫：モデル推論, 情報処理, Vol.32, No.3, 1991, pp.236–245.

[5] Kietz,J.-U. and Wrobel,S.: Controlling the Complexity of Learning in Logic through Syntactic and Task-Oriented Models, *Proc. of ILP-91,* 1991, pp.107–126.

[6] Muggleton,S. and Buntine,W.: Machine Invention of First-order Predicates by Inverting Resolution, *Proc. of Machine Learning 88,* 1988, pp.1–14.

[7] Muggleton,S. and Feng,C.: Efficient Induction of Logic Programs, *Proc. of the First Conf. on Algorithmic Learning Theory,* Ohmsha Ltd., 1990, pp.368–381.

[8] Quinlan,J.R.: Learning Logical Definitions from Relations, *Machine Learning,* 5, 1990, pp.239–266.

[9] Rouveirol,C.: ITOU: Induction of First Order Theories, *Proc. of ILP-91,* 1991, pp.127–157.

[10] 坂本忠昭：論理プログラムの並列帰納学習システム, 日本ソフトウェア科学会第8回大会論文集, 1991, pp.425–428.

[11] Shapiro,E.Y.: Inductive Inference of Theories from Facts, Technical Report 192, Yale University Computer Science Dept. 1981. （有川節夫訳：知識の帰納推論, 共立出版 1986）.