並列論理型言語上の制約充足方式の比較†

横尾 真†† 上田和紀†††

本論文では，従来から提案されている制約充足問題の解法である(1)単純なバックトラック型の解法，バックトラック型の解法の改良である(2)フォワードチェック型の解法と，近年提案された並列論理型言語向きの，処理の並列性を最大限に引き出すための新しい制約充足問題の解法である(3)レイヤードストリーム型の解法の比較を行う．レイヤードストリーム型の解法は並列に得られた途中解を併合して解を求めるという，従来の解法と非常に異なった性質を持つが，その処理量，並列度に関する考察が十分なされていなかった．本論文は実験的な評価と統計的モデルを用いた評価により，他の2方式との比較を行い，その性質を明らかにする．実験的な評価で得られた結果は統計的なモデルで示される性質とよく一致しており，以下の新しい結論が導かれた．(a)弱い制約が変数間に均一に存在する問題に対して，並列に得られた途中解を併合するレイヤードストリーム型の解法は，複数のプロセスが途中解を共有するため，1つの途中解を生成するための処理量が少ない．生成する途中解の個数は多いが，全体としての処理量は，バックトラック型の解法よりも少なく，フォワードチェック型の解法と同程度であり，処理の並列性は最も大きい．(b)一方，強い制約が一部の変数間にのみ存在する問題に対して，フォワードチェック型の解法は強い制約を早期に利用しうるが，レイヤードストリーム型の解法は，強い制約の存在により最終的な解の一部となりえない途中解を多く生成し，フォワードチェック型の解法と比較して処理量が多くなる．

1. まえがき

本研究の目的は，AIにおける重要な問題である制約充足問題[2]を，GHC[5]によって代表されるcommitted-choice型の並列論理型言語を用いて解く場合に，問題の特徴，解法により，全体の処理量，処理の並列性などにどのような違いが存在するかを発見することである．非常に多くの問題が制約充足問題として定式化可能であるが，現実的な問題では組合せ的に処理量が増大するのが普通であり，並列化による処理の高速化は重要な課題である．

並列論理型言語は，近年研究開発が精力的に行われている汎用的な言語である[5]．また，制約充足問題の解法としては従来から単純なバックトラック型の解法，バックトラック型の解法の改良であるフォワードチェック型の解法[2]などが知られていたが，並列論理型言語向きの，処理の並列性を最大限に引き出すための新しい制約充足問題の解法であるレイヤードストリーム型の解法が提案されている[4]．本研究ではこの新しい解法と，従来から存在するバックトラック型の解法，フォワードチェック型の解法について，これらの解法を並列論理型言語上で実現した場合の比較を行う．

以下，第2章では制約充足問題の定義と解法を示し，第3章では実際のプログラムにより，2つの異なる性質を持つ問題について行った実験結果を示す．第4章で統計的なモデルを用いて，実験で得られた結果の一般化を行う．

2. 制約充足問題の解法

2.1 問題の定義

制約充足問題は次のように定式化される．変数の有限集合 $I=\{x_1, x_2, \cdots, x_N\}$ および制約の集合が存在し，変数はそれぞれ値を値域 $D_1, D_2, \cdots, D_N$ からとる．I 中の k 変数間の制約 $c(x_{i_1}, \cdots, x_{i_k})$ は，直積 $D_{i_1}\times\cdots\times D_{i_k}$ の部分集合であり，互いに整合のとれた変数の値の組を表す．制約充足問題の解を求めることは，すべての制約を満たす変数の値の N 個組を求めることである．

例えば，N-queens は $N\times N$ の盤面に N 個のチェスのクイーンの駒を互いに取り合わないように並べる問題であるが，各列のクイーンの位置を表す N 個の変数の値を決定する制約充足問題として定式化できる．

2.2 並列型の解法

以下，まず従来のバックトラック型の解法を並列化した解法 (BT) とその改良版であるフォワードチェック型の解法 (FC) について説明し，次にレイヤードストリーム型の解法 (LS) について説明する．

2.2.1 バックトラック型の解法

従来のバックトラック型の解法，およびその改良版

† Comparison of Constraint Satisfaction Methods in Concurrent Logic Programming by MAKOTO YOKOO (NTT Communications and Information Processing Laboratories) and KAZUNORI UEDA (Institute for New Generation Computer Technology).

†† NTT 情報通信処理研究所
††† (財)新世代コンピュータ技術開発機構

であるフォワードチェック型の解法の並列化は容易である．以下，制約はすべて2変数間で定義されると仮定する．述語 $C_{ij}(x_i, x_j)$ は変数 x_i と x_j $(i<j)$ の間の制約を表し，変数 x_i の値 v_i と変数 x_j の値 v_j が制約を満たすときに，$C_{ij}(v_i, v_j)$ は真となる．バックトラック型の解法では，インクリメンタルに途中解*を生成して，制約を満たしているかどうかのチェックを行う．途中解は変数 $x_1, \cdots, x_k$ のすでに決定された値からなるリスト $V=[v_1, \cdots, v_k]$ で表される．解は手続き p-bt ([ ], 0) を呼ぶことによって得られる．

Procedure p-bt($[v_1, \cdots, v_k]$, k)

もし $k=N$ であれば $[[v_1, \cdots, v_k]]$ を返す．

さもなければ $v \in \{v \mid v \in D_{k+1}$ かつ $P_{k+1}(v_1, \cdots, v_k, v)\}$ なるすべての v に対して，手続き p-bt($[v_1, \cdots, v_k, v]$, $k+1$) を並列に実行し，結果のリストをつないだものを解とする．

単純なバックトラック型の解法 (BT) では，$P_{k+1}(v_1, \cdots, v_k, v)$ は，すべての $i<k+1$ に対して，$C_{i,k+1}(v_i, v)$ である場合に真となる．

一方，BT の改良である，フォワードチェック型の解法 (FC) では，より厳しいチェックを行うことにより，最終的な解となりえない途中解をより早く発見する．すなわち，$P_{k+1}(v_1, \cdots, v_k, v)$ は BT の条件に加えて，すべての $j>k+1$ に対して，$l \in D_j$ なる l が存在し，すべての $i<k+1$ に対して，$C_{ij}(v_i, l)$ が成立する場合に真となる．この条件は，途中解に対して，まだ値が決定されていないすべての変数について，少なくとも1つ制約を満たす値が存在することを要求する．フォワードチェック型の解法を実現する際には，通常，すでに決定された値のリスト $V=[v_1, \cdots, v_k]$ と共に，まだ決定されていない変数の領域のリスト $D=[d_{k+1}, \cdots, d_N]$ を持ち，変数 v_k の値を決定した時点で，いままで決定した変数の値との間の制約を満たさない値を D から取り除く操作を行う．

2.2.2 レイヤードストリーム型の解法

BT, FC は，途中解に対して，逐次的に新しい変数の値を1つずつ加えていくことにより解を求めている．一方，レイヤードストリーム型の解法 (LS) では，途中解に新しい変数を付け加えるのではなく，2つの途中解を併合することにより，新しい途中解を導く．

$S_1^1 = \{(v_1) \mid v_1 \in D_1\}, \cdots, S_N^1 = \{(v_N) \mid v_N \in D_N\}$

とし，手続き 1-s(1, $[S_1^1, \cdots, S_N^1]$) を呼ぶ．

Procedure 1-s(k, $[S_1^k, S_2^k, \cdots, S_{N-k+1}^k]$)

$k=N$ ならば，S_1^N を解として返す．さもなければ，$S_1^k, S_2^k, \cdots, S_{N-k+1}^k$ を用いて，

$$S_1^{k+1} = \{(v_1, \cdots, v_{k+1}) \mid (v_1, \cdots, v_k) \in S_1^k, (v_2, \cdots, v_{k+1}) \in S_2^k, C_{1,k+1}(v_1, v_{k+1}) \text{ が真}\},$$

$$\cdots,$$

$$S_{N-k}^{k+1} = \{(v_{N-k}, \cdots, v_N) \mid (v_{N-k}, \cdots, v_{N-1}) \in S_{N-k}^k, (v_{N-k+1}, \cdots, v_N) \in S_{N-k+1}^k, C_{N-k,N}(v_{N-k}, v_N) \text{ が真}\}$$

をそれぞれ並列に求め，1-s ($k+1$, $[S_1^{k+1}, \cdots, S_{N-k}^{k+1}]$) を呼ぶ．

例えば 4 queens では，$S_1^3 = S_2^3 = \{(1,4,2), (2,4,1), (3,1,4), (4,1,3)\}$，$S_1^4 = \{(2,4,1,3), (3,1,4,2)\}$ となる．S_1^3 と S_2^3 から S_1^4 を作る際には，すでに制約を満たしている値の組同士を併合するため，x_1 と x_4 の間の制約のみをチェックすればよい．ただし，このアルゴリズムを通常のリスト等を用いてインプリメントした場合，S_1^3 と S_2^3 から，x_2 と x_3 に同じ値を割り当てている組を選択する際に，S_1^3 と S_2^3 の要素のすべての組合せをチェックする必要がある．

レイヤードストリーム方式では，レイヤードストリームと呼ばれる再帰的なデータ構造を用い，S_1^3 と S_2^3 の要素の間でデータの共有を行うことにより，組合せのチェックを行わずに S_1^4 を計算することを可能にしている．具体的には，S_1^3 の要素 (2, 4, 1) と S_2^3 の要素 (4, 1, 3) は，x_2 と x_3 の値を表す (4, 1) の部分を共有しており，S_1^3 を検索することなく，(4, 1, 3) から (2, 4, 1) にアクセスすることができる．アルゴリズムの詳細については文献 4) を参照されたい．

3. 実験的評価

3.1 例　　題

2つの特徴的な例題に関して，実際のプログラムにより評価を行う．例題は 8 queens と5つの家の問題である．8 queens の特徴は，すべての変数の間に，比較的弱い制約があり，変数間の制約はほぼ均一であると考えられる点である．

一方，5つの家の問題 (five houses puzzle) は，制約が変数間で均一でない問題の例である．この問題では，5人の異なった国籍の人が同じ通りに建てられた，異なった色に塗られた5つの家に住んでいる，イ

* 途中解とは，変数の集合 $I=\{x_1, x_2, \cdots, x_N\}$ の部分集合 $I'=\{x_{i_1}, x_{i_2}, \cdots, x_{i_N}\}$ および，制約のうち I' 中の変数のみの間に定義されたものの集合についての制約充足問題の解を意味する．

ギリス人は赤い家に住む，ノルウェー人の住み家は緑の家の隣である等の事実が知られている．これらの事実を満たすように，各々の家の色，住む人の国籍等を見つけることが目的であり，異なる国籍，色等を表す25の変数に，家の番号1から5までを割り当てる制約充足問題として定式化できる．この問題では，強い制約（変数の値が等しいなど）がごく一部の変数の組合せの間にのみ存在し，平均的には変数間の制約は極めて弱い．

3.2 プログラム

BT, FC のプログラムは，Prolog で記述されたプログラムを，それぞれ文献6),7) の方法で人手により並列論理型言語のプログラムに変換したものを用いる．FC のプログラムは，領域をビットベクタで表現することにより，制約チェックに要する処理の回数を最適化したものを用いる．FC のプログラムの詳細は文献7)を参照されたい．LS のプログラムは，文献4)に示されている方法で記述したプログラムを用いる．

3.3 評価項目

これらの問題を解くプログラムに関して，実行時間，リダクション数，サイクル数，ノード数，制約チェック数を比較する．

実行時間は GHC のプログラムのコミットオペレータをカットオペレータに置き換えることによって Prolog の処理系で実行可能な形式としたプログラムで評価する．リダクション数，サイクル数は GHC の処理系のシミュレータで測定されたものである．リダクション数とは実行が終了するまでに行われたすべてのコミット操作の数であり，サイクル数とは，ゴールのうちコミットできるものをすべてコミットするという処理を1サイクルとした場合の，実行が終了するまでに繰り返されたサイクルの数である．

ノード数，制約チェック数は，前述のパラメータとは異なり，プログラムの書き方に依存しない値である．k 個の変数の値の組合せである途中解をレベル k のノードと呼び，それぞれの方式で作られるノードの個数を比較する．各レベルの異なるノードを計算する処理は，それぞれ全く独立に計算できるため，各レベルのノードの個数を，並列に実行できる処理の数を反映するパラメータと考えることができる．制約チェックの回数は，変数の値の組に対して，制約を満たしているかどうかのチェックを行う回数であり，次章で示す統計的モデルで，アルゴリズムの処理量を表すパラメータとして用いられる．

3.4 評価結果および考察

BT, FC, LS のそれぞれの実行時間，リダクション数等の測定結果をまとめて**表1**に示す．実行時間は，Sun 3/260 上の Quintus Prolog で測定した結果である．また，8queens，5つの家の問題のそれぞれについて，ノード数，制約チェック数の実測値を**図1, 2**に示す．ただし，FC は複数の制約のチェックを1回のビットベクタ操作で実現するが，図1, 2では，1回のビットベクタの操作を，それと等価な制約チェックの回数に置き換えた値を示す．これらの結果から示されることを述べる．

実行効率：8queens に関しては，実行時間は FC＜LS＜BT の順で，リダクション数は LS＜FC＜BT の順であるが，FC と LS の差はわずかである（表1）．一方，制約チェックの回数の合計は，FC＝12,560, LS＝11,780, BT＝40,282 となり，FC と LS の順序が逆転している．この理由は FC では複数の制約チェックが1回のビットベクタの操作で実現されているためと考えられる．FC では，低いレベルで多くのチェックを行うことにより，高いレベルでのチェック数が少なくなり，BT と比較してトータルのチェック数が少なくなっている．LS では低いレベルから，中間的なレベルにかけての制約チェックの回数が多い．この結果，FC と同様，BT と比較してトータルのチェック数が少なくなっている（図1）．

5つの家の問題に関しては，実行時間，リダクション数，制約チェックの回数の合計とも，FC＜BT＜LS の順となっており（LS の実行時間は計算機のメモリ不足のため測定が不可能となっている），方式による差が大きい．制約チェック数の合計は，FC＝27,356, LS＝2,238,441, BT＝90,455 となっている．8queens の場合と異なり，LS では中間的なレベルでの制約チェックの回数が極端に大きく，高いレベルでの制約チ

表1 実行時間，リダクション数，およびサイクル数の実測値
Table 1 Execution time and the numbers of reductions and cycles.

	Time (msec)	Reductions	Cycles
8 queens			
BT	6084	108759	172
FC	1110	20046	106
LS	1516	19418	38
Five houses			
BT	3067	53944	207
FC	1683	40061	472
LS	??	3153788	56

ェックの回数の減少と引き合わず，BT と比較してトータルのチェック数が多くなってしまう（図2）．この理由は，変数間の制約が均一でないため，互いの間の制約の弱い変数の組からなる途中解の個数が支配的となるためである．

処理の並列性：両方の問題について，LS のプログラムはリダクション数と比較してサイクル数が小さく，処理の並列度が最も大きい．制約を満たすノードの個数は FC<BT<LS の順となっており，LS は，各レベルでのノード数が多く，並列に実行できる処理の数が多いが，それぞれの処理の内容は1回の制約チェックのみであり，個々の処理の大きさは小さい．

並列論理型言語による特徴：プログラムの性質として，BT，FC のプログラムでは，生成されたプロセス間の通信はない．一方 LS では多くのプロセスがストリームによって通信を行う必要がある．LS のプログラムを並列マシン上で効率的に実行するためには通信のオーバヘッドを少なくする工夫が必要である．

また，BT, FC のプログラムをインプリメントする際には，並列論理型言語には破壊的代入の概念がないので，メモリの持っている破壊的代入機能を利用するために，末尾呼び出しの最適化 (tail recursion optimization) のような最適化技法に頼らざるをえない．このことが実行効率上デメリットとなることがある．一方，LS のプログラムをインプリメントする際には，論理変数の多重参照によってプロセス間でデータ共有ができることがメリットとなる．

4. モデルを用いた評価

本章では統計的なモデルを用いて前章で示した性質を一般化する．BT と FC に関する統計的なモデルが，文献 1), 3) に示されている．これらのモデルでは，制約充足問題を解くアルゴリズムの評価基準として，変数間の制約チェックの回数を用いており，その期待値を求めている．前章の結果も，アルゴリズムの処理量を大まかに予測するために制約チェックの回数を用

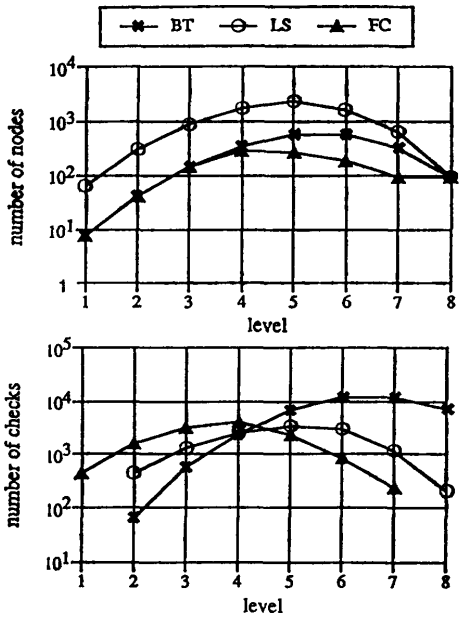

図 1 8queens でのノード数，制約チェック数の実測値
Fig. 1 Numbers of nodes and constraint checks of the 8 queens problem.

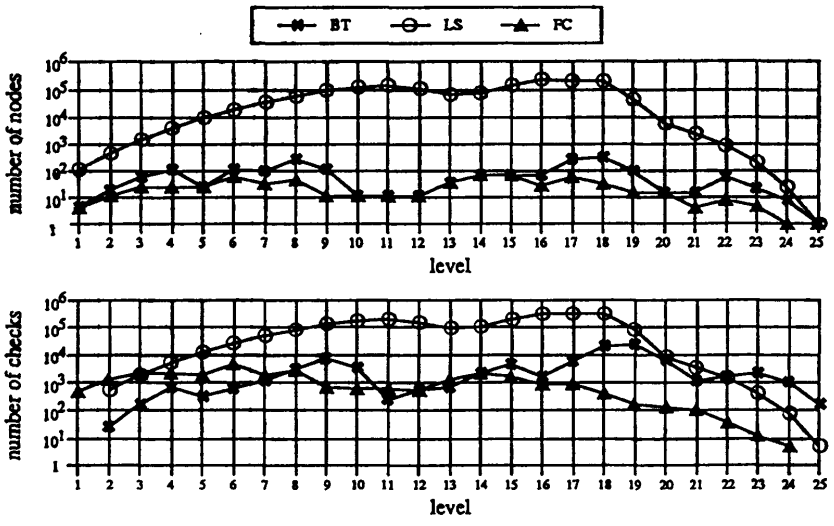

図 2 5つの家の問題でのノード数，制約チェック数の実測値
Fig. 2 Numbers of nodes and constraint checks of the five houses problem.

いることが可能であることを示している．

本章では以下，文献3)に示されている BT, FC のモデルを簡単に説明し，それらと同様な仮定に基づいた，新たに得られた LS のモデルを示す．これらのモデルを用いて，制約チェックの回数の期待値，各レベルのノード数の期待値に関して，例題について評価を行う．

4.1 統計的モデル

以下，次の仮定に基づいた統計的モデルを示す．

- 変数 x_i の領域の大きさは M_i で与えられる．
- 変数 x_i, x_j とそれぞれの値 v_i, v_j に対して，$C_{ij}(v_i, v_j)$ が成立する確率は一定の値 p_{ij} で与えられる（i, j によってのみ決まり，v_i, v_j には依存しない）．
- $C_{ij}(v_i, v_j)$ が成立する確率は，他の制約が成立する確率とは独立である．

4.1.1 バックトラックのモデル

レベル k のノード数は，

$$\left(\prod_{i=1}^{k} M_i\right)\left(\prod_{1\le l<m\le k} p_{lm}\right)$$

で与えられ，制約チェックの回数は

$$\left(\prod_{i=1}^{k} M_i\right)\left(\prod_{1\le l<m\le k-1} p_{lm}\right)p_{lm}\times\left(\sum_{i=1}^{k-1}\prod_{j=1}^{i-1} p_{jk}\right)$$

となる．式の×以降は，レベル $k-1$ のノード1つと，変数 x_k の1つの値との制約チェックの回数の期待値である．

4.1.2 フォワードチェックのモデル

制約を満たすレベル k のノード数は，

$$\left(\prod_{i=1}^{k} M_i\right)\left(\prod_{1\le i<j\le k} p_{ij}\right)\left(\prod_{f=k+1}^{N} S_f^{(k)}\right)$$

で，レベル k の制約チェックの回数は，

$$\left(\prod_{i=1}^{k} M_i\right)\left(\prod_{1\le i<j\le k} p_{ij}\right)\left(\prod_{f=k+1}^{N} S_f^{(k-1)}\right)\times\left(\sum_{i=k+1}^{N} m_{k-1}^{i}\prod_{j=k+1}^{i-1} S_j^{(k)}/S_j^{(k-1)}\right)$$

となる．ただし，

$$m_k^f = M_f\left(\prod_{1\le j\le k}^{k} p_{jf}\right)\Big/ S_f^{(k)}$$

$$S_f^{(k)} = 1-\left(1-\prod_{j=1}^{k} p_{fj}\right)^{M_f}$$

である．

4.1.3 レイヤードストリームのモデル

レベル k のノード数は，

$$\sum_{i=k}^{N}\left(\prod_{j=i-k+1}^{i} M_j\right)\left(\prod_{i-k+1\le l<m\le i} p_{lm}\right)$$

で与えられる．これは，$N-k+1$ 通りの変数の組合せの，レベル k のノードの個数を足し合わせたものである．x_{i-k+1} から x_{i-1} までの変数の組からなるレベル $k-1$ のノード1つに対して，このノードから派生するレベル k のノードを作るために必要な制約チェックの回数は，このノードの x_{i-k+2} から x_{i-1} までの変数の値と制約を満たす x_i の値の個数に等しい．このため，レベル k での制約チェックの数の期待値は，

$$\sum_{i=k}^{n}\left[\left(\prod_{j=i-k+1}^{i} M_j\right)\left(\prod_{i-k+1\le l<m\le i-1} p_{lm}\right)\times\left(\prod_{j=i-k+2}^{i-1} p_{ji}\right)\right]$$

で与えられる．

4.2 評価結果および考察

統計モデルの妥当性： 図3に，$N=8$, $M_i=M=8$ $(i=1,\cdots,N)$, $p_{ij}=p=0.65$ $(1\le i<j\le N)$ とした場合（この p の値は 8 queens の値に近い）の3つの解法の制約チェックの回数およびノードの個数の期待値を示す．得られた結果は図1の実験結果と定性的にはよく一致している．しかしながら，統計的モデルによる結果は，ノード数，制約チェック数とも，実測値よりも少なくなっている．例えば，図3の FC の統計的モデルによるノード数の期待値の合計は639個であるが，実際の 8 queens プログラムの生成するノード数の実測値は1,110個である．この原因は，制約が成立する確率が，他の制約が成立する確率と独立であるという

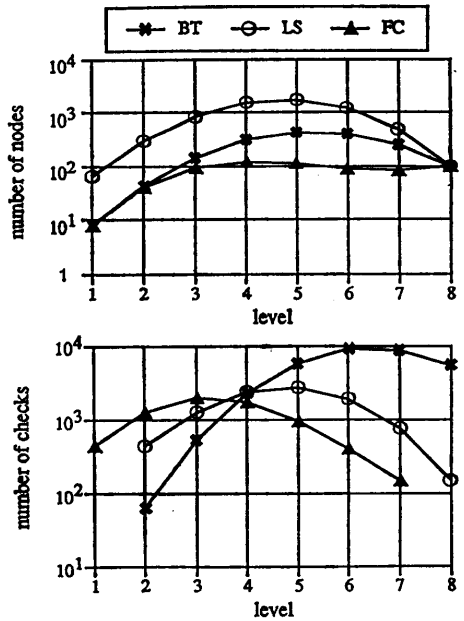

図3 $N=M=8$, $p=0.65$ でのノード数，制約チェック数の期待値

Fig. 3 Expected numbers of nodes and constraint checks in the case of $N=M=8$, $p=0.65$.

仮定にある．$p=0.65$ なる値は，最終的なノードの個数から求めたものである（$8^8\times p^{32}=92$ となるように選ばれたもの）．実際には 8 queens で個々の変数の値の組が制約を満たす確率は約 0.72 である．しかしながら，8 queens では，制約は独立ではなく（2つの制約は互いに独立であるが，3つ以上の制約の組合せは独立ではない），変数の値の組が複数の制約を満足する確率は，個々の制約を満足する確率の積よりも小さい．

制約が独立でない例として，{1, 2, 3} の permutation を作る問題を考える．この問題は，領域が {1, 2, 3} の A, B, C なる3つの変数があり，変数間には値が等しくてはいけないという制約がある制約充足問題として定式化できる．2変数間の制約は $p=2/3$ であるが，最終的な解の個数は $3^3\times p^3=8$ にはならず，実際には6である．3つの制約を満たす3変数の値の組の集合は，($A\neq B$ を満たす集合) $\cap$ ($B\neq C$ を満たす集合) $\cap$ ($C\neq A$ を満たす集合) である．もし，3変数の値の組の集合の部分集合である，$A\neq B$ を満たす集合のうち，$B\neq C$ を満たす要素の割合が 2/3 で，かつ，$A\neq B$ と $B\neq C$ を共に満たす集合中の，$C\neq A$ を満たす要素の割合が 2/3 なら，最終的な解の個数は8となるが，実際には，$A\neq B$ を満たす集合の 18 の要素中，$B\neq C$ を満たす要素の個数は 12 で確かに 2/3 であるが（$A\neq B$ と $B\neq C$ は独立），$A\neq B$ かつ $B\neq C$ なる集合 12 個の要素中，$C\neq A$ を同時に満たすものはその半分の6個しかなく，3つの制約は独立ではない．等しくてはいけないという制約は，有限の資源を複数の変数が取り合うことに近く，複数の制約を満たす確率は個々の制約を満たす確率の積よりも小さくなってしまう．

このような問題に対して，制約が独立であるという簡単化された仮定に基づく統計モデルを用いる場合には，p の値として，単純に個々の制約が成立する確率の平均を用いることはできず，種々の条件付き確率の平均を用いる必要があり，本論文で用いた最終的なノードの個数から p を求める方法は，平均的な確率を求めるための一方法である．8 queens で $p=0.72$ とおいた統計モデルによる結果は，最終的な解の個数，生成されるノード数，制約チェック数とも実際の問題よりもはるかに多く，8 queens の問題の性質を反映しない．$p=0.65$ の統計モデルによる結果は，おおむね 8 queens の問題の性質を反映しているが，解を求める途中の状態での制約の強さをやや強めに見積もっているため，ノード数，制約チェック数とも，実測値よりも少なくなっている．

変数の数，領域の大きさ，制約の強さの影響：**表 2** に，p を 0.65 に固定し，$N=M$ として N, M を変化させた場合の制約チェックの回数の比較を示す．制約チェックの回数は常に FC<LS<BT の順であり，N, M が増加するにつれ，これらの間の差が開いている．

図 4 に $N=M=8$ を固定して，p を変化させた時の制約チェックの回数の合計の比較を示す．p が小さい（制約が強い）場合には，BT<FC<LS の順となり，p が大きい（制約が弱い）場合には，FC<LS<BT の順となっており，LS と FC の差が小さい．制約が十分強ければ，BT でも急速に途中解の数が減少し，高

表 2 $N=M$, $p=0.65$ で N, M を変化させた場合の制約チェック数の期待値
Table 2 Expected number of constraint checks as a function of N, with $N=M$ and $p=0.65$.

N, M	BT	FC	LS
4	230	127	132
6	3,968	1,227	1,430
8	31,682	6,502	9,636
10	152,996	24,032	47,385
12	568,458	74,333	187,124
14	1,815,446	205,180	629,352
16	5,180,666	517,314	1,871,177

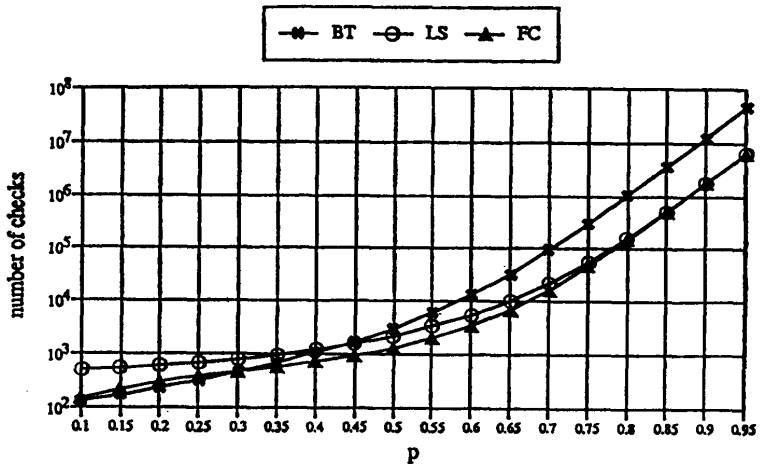

図 4 $N=M=8$ で p を変化させた場合の制約チェック数の期待値
Fig. 4 Expected number of constraint checks as a function of p, with $N=M=8$.

表 3 $N=25$, $M=5$ での制約チェックの回数の期待値
Table 3 Expected number of constraint checks for a non-uniform problem, with $N=25$, $M=5$ (50 permutations of variable ordering).

Algorithm	Minimum	Average	Maximum
BT	4.68×10^{6}	3.04×10^{11}	3.95×10^{12}
FC	1.80×10^{7}	4.08×10^{8}	5.64×10^{9}
LS	1.81×10^{6}	3.10×10^{10}	1.92×10^{11}

いレベルでの制約チェックの回数が減少するため，FC および LS の，低いレベルで多くの制約チェックを行うことが，より高いレベルでのノード数およびチェック数を減らすという効果が相対的に弱まってしまう．

変数の決定順序の影響：制約が均一でない場合には，BT の制約チェック数およびノード数は，変数を決定する順序に大きく依存する．5つの家の問題に近くなるように $N=25$, $M_i=M=5$ ($i=1,\cdots,4,6,\cdots,24$), $M_5=M_{25}=1$, p_{ij} を特定の8組の変数の間で 0.2（等しいに対応），特定の4組の変数間で 0.32（隣であるに対応），特定の 50 組の変数間で 0.72（1つの家には唯一の国籍，色等が割り当てられることに対応），その他の変数間では 1.0（制約チェックが必ず成功）に設定した場合の制約チェックの回数の合計の期待値は $FC=1.45\times10^{4}<BT=3.61\times10^{5}<LS=1.41\times10^{6}$ となる．この場合，BT, FC では，変数は $x_1,\cdots,x_{25}$ の順に決定されている．変数を決定する順序をこの逆の $x_{25},\cdots,x_1$ にした場合の制約チェックの回数の合計の期待値は，$FC=3.85\times10^{4}<LS=1.41\times10^{6}<BT=4.61\times10^{6}$ の順となる．LS では変数を決定する順序を逆にしても制約チェックの回数には影響がないが，BT では制約チェックの回数が増加し，LS との順序が逆転する．FC は BT と比較して順序の与える影響が小さい．

変数の値の決定順序の影響を見るために，表 3 に，$N=25$, $M_i=5$, p_{ij} は 15 組の変数の間で 0.2，その他の変数間では 1.0（制約なし）に設定し，ランダムに生成した 50 通りの変数の決定順序について，制約チェック数の平均，最大，最小の比較を示す．平均的には制約チェックの回数は FC<LS<BT の順となり，LS が BT よりも効率的となっている．しかしながら，BT, FC では，効率的な変数の決定の順序を求めるヒューリスティックスが研究されているが[3]，LS での効率的な変数の決定順序を求めることは今後の課題である．

5. むすび

本論文では，並列論理型言語上での3つの制約充足方式に関する比較を行った．実験的な評価に加えて，レイヤードストリーム型の解法について新しく統計的なモデルを示し，モデルに基づく比較を行い，レイヤードストリーム型の解法とフォワードチェック型の解法は，以下の性質を持つことが示された．

- ある途中解が，他の制約によって最終的な解の一部とならない可能性があまり大きくなければ，レイヤードストリーム型の解法は，複数の途中解を並列に計算し，途中解を併合して新しい途中解を生成することにより，バックトラック法と比べて全体の計算量を減らすことができる．
- 遠隔の変数（$i-j$ が大きいような x_i と x_j）間の制約があるとき，レイヤードストリーム型の解法では，この制約はすぐには利用されない．近接の変数間の制約が多いときにレイヤードストリーム型の解法は有効であり，遠隔の変数間に強い制約があり，近接の変数間の制約が弱い場合は，強い制約によって最終的な解となりえない途中解を多く生成する可能性が大きい．
- フォワードチェック型の解法では，遠隔の変数間の制約も早期に利用されうる．それらの間に強い制約があるときは特に有効である．

今後の課題として，通信のオーバヘッドも含めたアルゴリズムの実行効率を評価するために，並列計算機上で実行時間に関する評価を行うことが挙げられる．

謝辞 本研究の基本的なアイデアは横尾の ICOT 滞在中に得られたものである．滞在の機会を与えてくださった ICOT の淵所長，神奈川大学の村上教授（当時 NTT 情報通信処理研究所知識処理研究部部長）に感謝します．また，熱心に御討論頂いた ICOT の古川研究担当次長，長谷川研究部長代理（当時 ICOT 第一研究室室長），評価用のプログラムを提供して頂いた沖電気の奥村氏（当時 ICOT 第一研究室）に感謝します．

参考文献

1) Haralick, R. M. and Elliot, G. L.: Increasing Tree Search Efficiency for Constraint Satisfaction Problmes, *Artif. Intell.*, Vol. 14, pp. 263-313 (1980).
2) Mackworth, A. K.: Constraint Satisfaction, in Shapiro, S. C. (ed.), *Encyclopedia of Artificial Intelligence*, John Wiley & Sons, pp. 205-

211 (1987).
3) Nudel, B.: Consistent-Labeling Problems and Their Algorithms: Expected-Complexities and Theory-based Heuristics, *Artif. Intell.*, Vol. 21, pp. 135-178 (1983).
4) Okumura, A. and Matsumoto, Y.: Parallel Programming with Layered Streams, *Proc. 1987 Symposium on Logic Programming*, IEEE, pp. 224-231 (1987).
5) Ueda, K.: Guarded Horn Clauses, Tech. Report TR-103, ICOT, Tokyo, 1985. A revised version in *Logic Programming '85*, LNCS 221, pp. 168-179, Springer-Verlag (1986).
6) Ueda, K.: Making Exhaustive Search Programs Deterministic, *Proc. Third Int. Conf. on Logic Programming*, LNCS 225, pp. 270-282, Springer-Verlag, 1986. A revised version in *New Generation Computing*, Vol. 5, No. 1, pp. 29-44 (1987).
7) Yokoo, M. and Ueda, K.: Solving Constraint Satisfaction Problems in Concurrent Logic Programming (in Japanese), *ICOT Tech. Memorandum*, ICOT, 1988. Also in 日本ソフトウェア科学会知識プログラミング研究会第3回ワークショップ, pp. 1-7 (1988).

(平成2年 9 月 10 日受付)
(平成2年 12 月 18 日採録)

横尾 真 (正会員)

1962年生. 1984年東京大学工学部電子工学科卒業. 1986年同大学院修士課程修了. 同年 NTT に入社. 現在 NTT 情報通信処理研究所, 知識処理研究部勤務, 知識処理に関する研究に従事. 分散 AI, 制約充足問題等に興味を持つ. 人工知能学会, ソフトウェア科学会各会員.

上田 和紀 (正会員)

1978年東京大学工学部計数工学科卒業. 1986年同大学院情報工学博士課程修了. 工学博士. 1983年日本電気(株)入社. 1985年6月より(財)新世代コンピュータ技術開発機構に出向中. プログラム言語, 並行・並列処理, インタラクティブ・システムに興味を持つ. ACM, 日本ソフトウェア科学会各会員.