博士学位論文

ニューラルネットワークを用いた制約充足問題及びその拡張問題の解法に関する研究

中野　隆宏

九州工業大学大学院　生命体工学研究科　脳情報専攻

2006年3月

目次

第1章　序論

現代の高度情報化社会において，コンピューターの重要性はますます高まっている．技術の発展により，計算機に搭載されるCPUの動作クロックは向上し，近年はより高速な演算能力を備えた計算機が利用可能となった．これらの計算機はノイマン型の計算原理に基づいており，その計算原理において入力サイズの多項式時間で解くことができる問題のクラスをクラスPという．クラスPに属する問題としては，最短路問題，最大流問題，線形計画問題などが挙げられる．一方，非決定性アルゴリズム（選択分岐に直面したならば，それぞれの分岐枝に計算機を割り当て処理を続行することのできる仮想的なアルゴリズム）により多項式時間で解けるクラスの問題をクラスNPという．クラスNPに属する全ての問題をある問題に多項式時間で帰着することができるとき，その問題をNP困難問題という．またNP困難問題の中でクラスNPに属する問題をNP完全問題という．つまり，NP完全/困難問題を効率的に解くアルゴリズムを開発したならば，クラスNPに属する問題全てを効率的に解くことができる．工学の応用分野において，多くの重要な問題のほとんどがNP完全/困難問題である．例えば，巡回セールスマン問題，集合分割問題，ナップサック問題，ハミルトン閉路問題，充足可能性問題（SATisfiability problem; SAT）などがNP完全/困難問題として知られている．

これまで多くの研究者がNP完全/困難問題を効率的に解くアルゴリズムの発見に挑戦してきたが，現在はそのようなアルゴリズムは存在しないのではないかと予想されている．そのため近年では，厳密な解法で解くには非常に時間かかってしまうような困難な問題に対しては，効率的に解くことができる問題のサブクラスを見つける，ある質の近似解が保証されているアルゴリズムを適用する，試行錯誤的に解を探索する発見的手法を適用するといったアプローチが実用的にはとられている．ノイマン型のアルゴリズムではこのように越えることのできない計算量の壁が存在する．そしてノイマン型のアルゴリズムは効率性の問題とともに柔軟性に欠けるという問題を持っている．従来のノイマン型のアルゴリズムでは設計者が設計した手続きに従って動作する．換言すればアルゴリズムは決めらた手続き通りにしか

動かないので，少しでも想定していない入力が与えられた場合，アルゴリズムは正しく動作しない．設計者はあらゆる場合を想定して複雑なプログラムを作る必要がある．そのため，近年では，ノイマン型に変わる柔軟で高速かつ超並列実行可能な脳型計算原理の実現を目指した研究が注目されており，ニューラルネットワークを用いた計算原理[1]などが提案されている．新たなる計算原理により，脳型計算機が実現できたならば，ノイマン型計算機で問題であった効率性や柔軟性の問題を克服できると考えられる．

前述したようにNP完全問題の代表的なものとしてSATがある．SATとは与えられたCNF論理式（Conjunctive Normal Form）を真とするような変数割当が存在するか否かを判定し，解が存在するならば解を見つける問題である．我々は，SATの解法としてLPPH（Lagrange Prgramming neural network with Polarized High-order connections）[2]~[5]と呼ばれるニューラルネットワークを用いた手法を提案している．この手法では，まず0，1の離散的な組合せ問題であるSATをそれと等価な連続的な問題であるCONSAT(CONtinuous valued SATisfiability problem)に変換する．このCONSATに対して各変数が0から1の間の実数値をとるニューラルネットワークであるLPPHを適用する．組合せ最適化問題を解くニューラルネットワークとしてはホップフィールドニューラルネットワークが広く使われている．しかし，ホップフィールドニューラルネットワークはエネルギー関数の勾配方向に変数を更新していく手法であるため，局所最適解（解ではないが近傍によりよい状態がないため変数を更新できない状態）に陥るという問題がある．一方，LPPHではCNFの節の非充足度関数（充足しているなら0，そうでないなら正の値を返す関数）と重みの積の総和であるラグランジュ関数を考える．このラグランジュ関数において重みを固定するとエネルギー関数として捉えることができる．LPPHでは変数に対応するニューロンはラグランジュ関数の勾配を降りる方向で変化し，重みは勾配を登る方向へ変化する．そのため，LPPHは局所最適解に陥ることなくSATの解を見つけることができる．実験結果よりLPPHはGSATなどのSATの既存解法に比べ高速に解を見つけ出すことが確認されている．

LPPHにおいて節の非充足度関数はその節に含まれるリテラルの非充足度の乗算により計算される．しかし，乗算による定義では現在の変数割当が節を充足しているかどうかについて，直観的な評価とは異なる評価を行ってしまう場合がある．この問題は直観的な評価に即した非充足関数を再定義する[6]ことにより克服することができる．新たな節の非充足度関数はMIN演算により定義される．さらに我々はLPPHを制約充足問題（Constraint Satisfaction

Problem;CSP）を解くために拡張を行った[6]~[8]．CSPとは，与えられた制約を全て満たすような変数への値の割当を探索する問題であり，その表現の抽象性から輸送計画問題，生産計画問題，資源割当問題，基盤設計問題，時間割作成問題，スケジューリング問題などの実社会での実際に現れる組合せ問題を表現することができる．SATもCSPの一種であり，CNF論理式の節を制約とみなすと，SATは全ての制約を満たす（節を真にする）ように変数へのブール値の割当を探索するCSPとして捉えることができる．我々は，MIN/MAX演算を制約充足問題の制約に対する非充足度関数の定義に用いることによりLPPHの力学系の拡張を実現した（この力学系をLPPH-CSPと呼ぶ）．

CSPの解法は大きく二分すると完全解法[9],[10]と不完全解法[11]~[14]に分けることができる．完全解法とは，変数に順に値を割り当てていき，問題の制約と矛盾したならばバックトラックすることにより解を探索する木探索法[10]に代表される系統的探索アルゴリズムである．完全解法では，最悪の場合，全組合せを試すことになるので，アルゴリズムを動かし続ければ，いずれは解を見つけること，または問題に解がないことを判定することが保証されている．しかし，一般に完全解法は問題の規模が大きくなると解の探索時間が膨大なものになってしまう．一方，不完全解法は全ての変数に初期値を割り当て，制約違反が少なくなるように変数を更新していく局所探索法に代表されるアルゴリズムであり，完全解法とは異なり，解の存在を判定することはできない．しかし，問題に解が存在する場合，完全解法に比べ高速に解を見つけ出すことができる．そのため，問題の規模が大きい実問題での応用を考えた場合，高速な不完全解法の適用が現実的であり，多くの研究者が高速に解を探索するために様々な不完全解法のアルゴリズムを開発している．

従来，CSPやSATのような離散的な組合せ問題の解法としては，離散的なアルゴリズムを適用するのが一般的であった．SATの研究者であるGuはSATを連続的な問題であるUniSAT（Universal SATisfiability problem）で表し，そのUniSATを解くために，離散値をとる幾つかの解法を提案している[15],[16]．しかし，Guは実験結果から連続値を値としてとり連続的な解空間を探索する解法は，離散値を値としてとり離散的な解空間で解を探索する解法に比べ，効率的でないと結論付けている．しかし，我々が提案したLPPHは連続値をとる解法であり，かつ効率的に解を見つけることが実験により確認されている．CSPの効率的な解法として知られているGENET[12],[13]は，離散値をとる解法であり，そのため全ての変数の値を同時に更新すると，解の振動現象を起こすため解を見つけることができなくなってしま

う[13]．そのため，変数は一つずつ逐次に更新する必要がある．LPPHの電子回路による実現はすでに行われているが，LPPHと同様にLPPH-CSPの電子回路による実現ができると，変数の値を同時に更新しながら，効率的にCSPの解を見つけ出すことができ，種々の問題の超並列解法として大いに意義があるといえる．

組合せ問題によく現れるような一般的な制約を定義し，その制約を用いて問題を表現することにより，従来CSPの分野でよく使われる2項制約や，SATのCNF論理式における節のみを用いた表現では，膨大な制約数を必要とする問題をコンパクトに表現することができる．そして，そのコンパクトな表現で制約充足を行うことにより，従来の膨大なサイズの表現に対する解法よりもより早く解を見つけ出すことが期待できる．このことに関して，制約充足問題を解く際の制約の記述方法が探索に大きな影響を与えることを本論文では明らかにした．さらに各制約の重要度をLPPH-CSPに導入することを提案し、その有効性を示す[17]．CSPはSATなどに比べ，より効率的に問題を記述することができるが，実問題への適用を考える場合，満たすべき制約とともにコスト最小化などのなんらかの目的関数を必要とする場合がある．そこで，本論文では，目的関数を持つCSP(OCSP)を定義し，その問題を解くために，LPPH-CSPに目的関数を考慮した項を追加した力学系LPPH-OCSP[18]を提案する．

本論文では，2章にCSPの定義と組合せ問題によく現れるような一般的な制約の定義を行う．またCSPで表現することができる問題を例とともに示し，その問題が一般的な制約を用いることにより，どのようにCSPで記述されるかを示す．3章では，代表的なCSPの完全解法と不完全解法の紹介を行なう．本論文では連続値によるCSPの不完全解法であるLPPH-CSPを提案するが，これまでに連続値の解法としてはどのようなものが提案されてきたかを紹介する．4章ではSATを解くニューラルネットワークであるLPPHをその性質とともに示す．また，LPPHの非充足度関数における問題点を解消するために新たな定義式を示し，その有効性を実験結果とともに議論する．5章ではCSPを解くためにLPPHの拡張を行なう．CSPの制約に対する非充足関数の定義式として3つの手法を示し，その有効性を従来のLPPHによる解探索とともに論議する．そして，LPPH-CSPを提案し，その性質と利点を明らかにし，CSPの効率的な解法であるGENETとの比較実験を行なう．また，問題記述に冗長な制約を付加した場合，より早く問題の解を見つけることができる場合があることを示す．さらにCSPの各制約に対して重要度を考慮したLPPH-CSPを提案する．6章では目的関数付きCSP(OCSP)を新たに定義し，その問題を解くためにLPPH-CSPを拡張す

る. OCSP を解くために力学系において導入した新たな項の役割を実験とともに示し，提案手法を 0-1 整数計画問題の解法である lp_solve，OPBDP [19] と比較し，その有効性を検証する. 最後に 7 章で本論文の結論を示し，今後の研究課題について述べる.

第2章　制約充足問題

2.1　制約充足問題の定義

制約充足問題 (Constraint Satisfaction Problem; CSP) とは，与えられた制約を全て満足するように変数に値を割り当てる問題である．制約充足問題は 3 項組 (X, D, C) として次のように定義される．

- X は変数の有限集合．
- D は離散値を要素とする変域の有限集合であり，変数 X_i には変域 D_i の要素である値が一つだけ割り当てられる．
- C は制約の集合．C の各制約は変数がとりうる変域の要素の値を制限する．

制約の集合 C を全て満たすような変数の集合 X への値の割当が CSP の解となる．

2.2　制約充足問題で用いる制約

従来，CSP の制約として広く使われているのが 2 項制約である．これはとってはならない 2 つの変数の組み合わせを禁止する制約であるが，2 項制約のみで複雑な組合せ問題を表現する場合，制約数が膨大なものになってしまう．そこで本節では組合せ問題によく現れる制約を一般化し，定義する．本節で述べる一般的な制約を用いることにより，2 項制約で問題を表現する場合に比べ，問題を表現する際に必要とされる制約数を少なくすることが可能になる．一般的な制約を説明する準備として，“変数 X_i が変域 D_i の中で j 番目の値をとる”ことを表す変数を x_{ij} とし，これを VVP [13] と呼ぶ．$x_{ij} = 1$ ならば，“変数 X_i が変域 D_i の中で j 番目の値をとる”ことが成り立ち，$x_{ij} = 0$ ならばそれが偽となる．この VVP x_{ij} を用いて一般的な制約を次のように定義する．

- ALT(n, S) [at-least-n-true constraint]

S は VVP の有限集合である．ALT(n, S) は S の中で少なくとも n 個の VVP が真となることを要求する．

- ALF(n, S) [at-least-n-false constraint]

  ALF(n, S) は S の中で少なくとも n 個の VVP が偽となることを要求する．

- AMT(n, S) [at-most-n-true constraint]

  AMT(n, S) は S の中で多くとも n 個の VVP が真となることを要求する．

- AMF(n, S) [at-most-n-false constraint]

  AMF(n, S) は S の中で多くとも n 個の VVP が偽となることを要求する．

例として，頂点彩色問題を上記の制約を用いて表現する．N 色の頂点彩色問題は与えられた無効グラフの全てのノードを N 色の色を用いて，隣接するノードを同じ色に塗らないように彩色する問題である．図 2.1 に 2 色（黒と白）の頂点彩色問題の解の例を示す．そしてこの問題を CSP を用いて表現したものを図 2.2 に示す．図 2.2 において，X_1 はノード 1 の色を表し，x_{11} は “ノード 1 が色 1 によって塗られる” ことを表している．また，ALT 制約 (C_1, C_2, C_3) と AMT 制約 (C_4, C_5, C_6) は “各ノードがちょうど 1 色の色で塗られる” ことを表す．そして，ALF 制約 (C_7, C_8, C_9, C_{10}) は “隣接するノードは同じ色では塗られない” ことを表現している．

一般的に AMT$(1, \{x_{11}, x_{12}, \cdots, x_{1N}\})$ を 2 項制約で表現する場合，$\{x_{11}, x_{12}, \cdots, x_{1N}\}$ 中の任意の 2 組の VVP が同時に成り立つことを禁止することにより表現する．つまり，2 項制約では，1 つの AMT$(1, \{x_{11}, x_{12}, \cdots, x_{1N}\})$ を表現するために ${}_NC_2$ 個の制約を必要とする．このように上記の一般的な制約を用いることにより，コンパクトに問題を表現することができる．

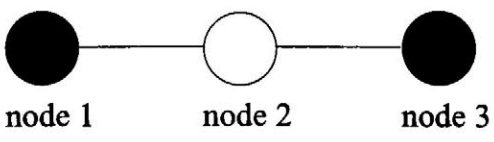

図 2.1　2 色の頂点彩色問題の解

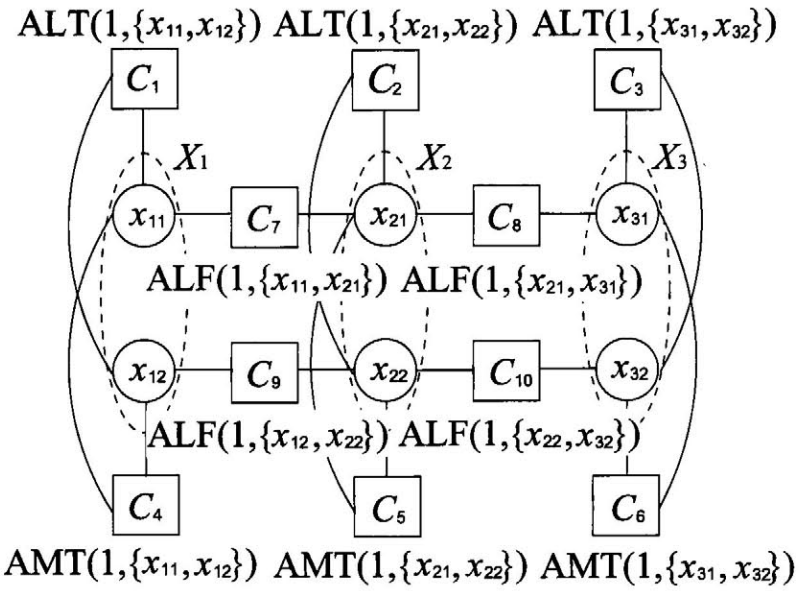

図 2.2　2 色の頂点彩色問題を CSP により表現

2.3　制約充足問題で表現できる問題

2.3.1　充足可能性問題

充足可能性問題 (SATisfiability problem; SAT) とは与えられた CNF 論理式を真とするような変数の割当を探索する問題であり，制約充足問題の一種である．$X = \{x_1, x_2, \cdots, x_n\}$ をブール変数の集合とすると，CNF 論理式は次式のような節の論理積によって定義される．

$$E = C_1 \wedge C_2 \wedge C_3 \wedge \cdots \wedge C_m. \tag{2.1}$$

CNF 論理式の節 C_r はリテラルの論理和によって表現される．

$$C_r = L_{r1} \vee L_{r2} \vee \cdots \vee L_{rl_r}. \tag{2.2}$$

ここでリテラルとは変数もしくはその否定である．SAT は次のように定義される．

$$\begin{aligned} &\text{(SAT)} \quad \text{find } \boldsymbol{x} \text{ such that} \\ &\qquad \boldsymbol{x} \text{ satisfies } C_r,\ r = 1, 2, \cdots, m, \\ &\qquad \boldsymbol{x} \in \{0,1\}^n. \end{aligned} \tag{2.3}$$

CNF 論理式の節 C_r は "C_r に現れるリテラルのなかで少なくとも 1 個のリテラルが真である" という制約として捉えることができる．つまり，SAT とは，各変数 x_i が変域 $D_i = \{0, 1\}$ を値としてとり，CNF 論理式により問題を表現する CSP である．

2.3.2 Car Sequencing Problem

car sequencing problem[20] はスケジューリング問題の一種であり，与えられた制約を満足するように車の生産スケジュールを決定する問題である．この問題ではラジオ，サンルーフ，エアコンといった幾つかのオプションを車に付けることにより車をタイプ分けし，それぞれのタイプを要求された台数だけ生産する．生産現場では，車はベルトコンベアーにのり，異なったエリアで各オプションはそれぞれ付けられる．生産ラインの各エリアでは，技師が許された時間内にオプションを付ける仕事をする．例えば，サンルーフを付けるには 20 分の時間を要する．しかし，車はベルトコンベアーにのって 4 分毎にくるため，そのエリアでのオプションの付けることができる車の最大の容量は 5 台ということになる．生産ラインでは異なったタイプの車を混在して作ることが要求され，各タイプで要求される車の台数は生産制約として与えられる．車のタイプは付けるオプションの種類によって差別化されれる．例えば，あるタイプではエアコンとパワーブレーキが付いており，残りのオプションは付いていない．car sequencing problem の解は与えられた生産制約と容量制約を満たすような車へのオプション付加の順序付けである．car sequencing problem の例を表 2.1, その解の 1 つを表 2.2 に示す．表 2.2 において，生産する車のタイプの順番が問題の解である．

表 2.1　car sequencing problem 問題の例

	車のタイプ						
オプション	type1	type2	type3	type4	type5	type6	容量制約
A	1	0	0	0	1	1	1/2
B	0	0	1	1	0	1	2/3
C	1	0	0	0	1	0	1/3
D	1	1	0	1	0	0	2/5
E	0	0	1	0	0	0	1/5
生産制約	1	1	2	2	2	2	

表 2.2　car sequencing problem の解

ライン上の車	車のタイプ	オプション				
		A	B	C	D	E
car1	type1	1	0	1	1	0
car2	type2	0	0	0	1	0
car3	type6	1	1	0	0	0
car4	type3	0	1	0	0	1
car5	type5	1	0	1	0	0
car6	type4	0	1	0	1	0
car7	type4	0	1	0	1	0
car8	type5	1	1	0	0	0
car9	type3	0	1	0	0	1
car10	type6	1	1	0	0	0

car sequencing problem の難しさは平均利用率を計算することによって知ることができる．利用率とは生産制約によって要求されるオプション i が付加された車の台数と容量制約が許容する最大のオプション i の付加された車の台数との比を意味する．たとえば，表 2.1 の問題においてオプション A が付加される車の台数は生産制約より 5 台である．一方，容量制約によって許容されるオプション A の付加された車の最大台数は 5 台である．よってオプション A の利用率は 100%となる．平均利用率とは全てのオプションの利用率の平均である．平均利用率が高くなるにしたがって問題は難しくなる．

2.3.3　N-Queen 問題

N-Queen 問題とは $N \times N$ の盤目からなるチェスの盤面上に N 個の Queen の駒をお互いに取られないように配置する問題であり，組合せ問題のベンチマーク問題として広く知られている．Queen の駒は図 2.3 で示されるようにチェスの盤面上で縦横ななめ方向に好きなだけ移動できる．

$N \times N$ の盤上にお互いに取られないように N 個の Queen を置くためには，次のような制約が要求される．

- 各行には必ず 1 つ Queen がある．

- 各列に多くとも 1 つしか Queen を置けない.
- 各右上がり斜め方向に多くとも 1 つしか Queen を置けない.
- 各左上がり斜め方向に多くとも 1 つしか Queen を置けない.

上記の制約を満たすような変数の割当が N-Queen 問題の解となる．例として，9-Queen 問題の解の 1 つを図 2.4 に示す.

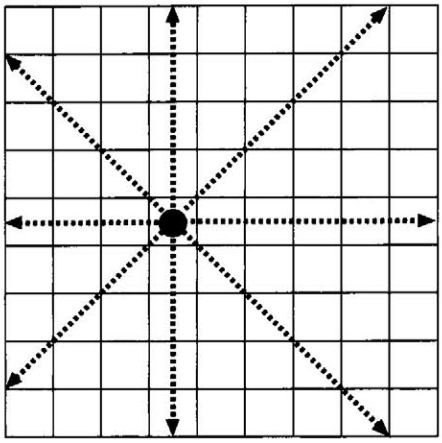

図 2.3　Queen の動き

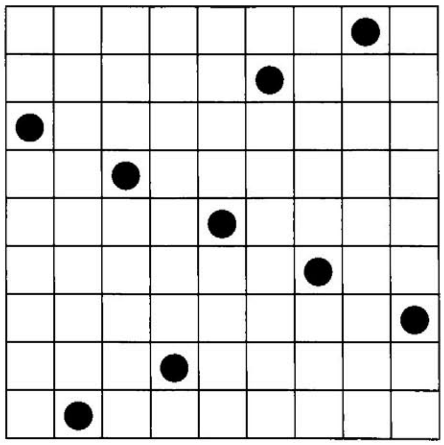

図 2.4　9-Queen 問題の解

2.3.4 All-Interval-Series Problem

all-interval-series problem とは serial music composition において起こる有名な問題に想起された問題である[21]. 12のスタンダードピッチクラス（c, c#, d, ···）が与えられたとき，このクラスをあらわす数0, 1, ···, 11において各ピッチがちょうど1つ使われる．また，各ピッチの間隔の集合が，マイナーセカンド（1セミトーン）からメジャーセブン（11セミトーン）をカバーするように割り当てられる．この問題は集合 Z_n におけるより一般的な演算問題の事例としてとらえることができる．all-interval-series problem とは与えられた正の整数 n において，次のようなベクトル $s = (s_1, s_2, \cdots, s_n)$ をみつける問題である．

- $s = (s_1, s_2, \cdots, s_n)$ は $Z_n = \{0, 1, \cdots, n-1\}$ の順列.
- 間隔ベクトル $v = (|s_2 - s_1|, |s_3 - s_2|, \cdots, |s_n - s_{n-1}|)$ は $Z_n - \{0\} = \{1, 2, \cdots, n-1\}$ の順列.

上記のようなベクトル $s = (s_1, s_2, \cdots, s_n)$ は次のような制約によって表現することができる．

- 各 s_i は $Z_n = \{0, 1, \cdots, n-1\}$ から1つだけ値をとる.
- Z_n の各値は1回だけしか選ばれない.
- 各 v_i は $Z_n - \{0\} = \{1, 2, \cdots, n-1\}$ から1つだけ値をとる.
- $Z_n - \{0\}$ の各値は1回だけしか選ばれない.
- s_i かつ s_{i+1} ならば v_i となる.

2.3.5 時間割作成問題

大学などの教育機関における時間割は教官や教室の都合によって必要とされる制約を満たすように作成される．従来は，人の手によってこのような時間割は作成されていたが，規模が大きくなると制約を満たすような時間割を作成するのは大変であり，制約充足問題の解法を適用することの重要性が大きくなる．大学の時間割を作成する場合，次のような情報を考慮しければならない[22].

1. 生徒の集合 $\sum_S$，教授の集合 $\sum_P$，教室の集合 $\sum_R$，時間枠の集合 $\sum_T$，講義の集合 $\sum_C$.
2. 各生徒がどの講義を希望するか. 例えば生徒 s はコース c_1，c_2，c_3 を受講することを希望する.

3. 各教授はどの講義を教えるか.

4. 各教授が教えることはできない時間枠.

5. 各講義が使うことができない教室（収容人数の問題など）.

6. 各教室が使うことができない時間枠.

時間割作成問題では，上記の情報を用いて次のような制約を満たす解を探索する.

（a）時間割は全ての講義を含む.

（b）各時間枠と各教室において少なくとも 1 つの講義が行なわれる.

（c）もし，2 つの講義 c_1，c_2 が同じ教授によって教えられるならば，c_1 と c_2 は異なった時間枠に割り当てなければならない.

（d）2 つの講義 c_1，c_2 を同じ生徒が希望するならば c_1 と c_2 は異なった時間枠に割り当てなければならない.

（e）もし講義 c_1 が時間枠 t_1 に教えることができない教授によって教えられるならば，t_1 に c_1 を割り当ててはならない．5. と 6. の情報も同様にして制約として与えられる.

2.4 制約充足問題の難しさ

CSP では，どのような問題が難しく，どのような問題が簡単だろうか？ 一般的に，完全解法と不完全解法の両方において，問題の変数の数などのサイズが大きい場合には，それだけ探索の時間がかかってしまう．それ以外にも問題の難しさを測る基準として変数と制約数の比が挙げらる．多くの研究者によってどのような比が最も難しいか，またなぜその比が問題を難しくしているかについて研究がなされている．Mitchell らは SAT の有名な完全解法である DP [23] を random 3-SAT に適用した際に，変数の数に対する制約の数の比が 4.3 付近になる問題が最も解くのに時間がかかることを示した [24]．制約と変数の比が 4.3 付近になるように random 3-SAT を生成した場合，その問題に解が存在する確率はおよそ 50%であることが分かっている．つまり，解の存在する確率が 50%付近になるようなパラメータで生成された問題が最も難しい．そしてその前後で解の存在する確率が急激に変化することが明ら

かになっている．この変化は相転移と呼ばれ，組合せ最適化問題の難しさを測る上で大きなキーワードとなっている．

一般に完全解法では，変数に対して制約が緩い場合，容易に制約を満足するような解を見つけることができる．一方，変数に対して制約が多く与えられる問題（解の存在する確率が低い問題）に対しては，問題の探索空間の大部分は枝刈りすることができ，そのため，解を容易に発見することができる，もしくは，解がないことを容易に判定することができる．つまり，解が存在する確率が 50%付近では，上記の中間に位置するため，最も難しくなるとされている．また横尾は不完全解法でも相転移付近の問題が最も難しくなることを山登り型探索アルゴリズムを用いた実験により明らかにした[25]．制約が緩い場合は不完全解法と同様に山登り探索アルゴリズムに代表される不完全解法でも容易に解を見つけることができる．そして，相転移付近よりも解の存在確率が低くなるような問題の方が易しくなる理由は局所最適解の個数によって説明されている．山登り型探索アルゴリズムでは，制約を増やすと解の個数は減少するため問題は難しくなるが，同時に局所最適解の個数も減少する．また，制約を付加すると山登り探索アルゴリズムにおいて解の評価値は 0 のままだが，局所最適解の評価値は増加する．そのため，解及び解に到達可能な状態の評価値の方が局所最適解の評価値よりも相対的に小さくなるため，解への到達可能性が大きくなる．さらに，解の個数は相転移付近で急減に減少するが，局所最適解の個数はほぼ一定の比率で減少している．つまり制約が多い場合は，局所最適解の数の減少による問題の簡単化が解の減少による難易度増加を上回っているため，相転移付近よりも易しくなっているとされている．

あらかじめ問題の難しさを測ることができるならば，問題を解く際の解法の選択する基準の 1 つとして考えることができる．そして問題の難しさや何が問題を難しくしているかを解析することは，効率的な解探索を実現するアルゴリズムの開発にも重要な示唆を与える．このような理由から相転移をはじめとする問題の難しさの解析をする研究は組合せ最適問題の分野において重要な意義を持つ．

第3章　制約充足問題の解法

CSPの解法は大きく二分すると完全解法と不完全解法に分けることができる．木探索法[10]に代表される系統的探索アルゴリズムである完全解法は，最悪の場合，全ての変数の値の組合せについて調べるので，与えられた問題が解が存在するどうかを判定できることが保証されている．しかし，完全解法は問題のサイズが大きくなると，解探索に要する時間は膨大なものになる．山登り型探索アルゴリズムなどの局所探索法に代表される不完全解法は，最初に全ての変数になんらかの方法で値を割り当て，ある評価にしたがってその値を変えていくことにより，解を探索する．完全解法とは異なり，不完全解法では解が存在しないこうとを判定をすることはできないが，与えられた問題に解が存在する場合，一般に完全解法に比べ，短い時間で解を発見することができることが知られており，組合せ最適化の分野では近年，広く研究がなされている．

3.1　制約充足問題の完全解法

一般的に木探索法が完全解法として使用できるが，それ以外の方法として併合法[9]と呼ばれる方法がある．併合法は，可能な値の組合せを表す制約をノードに，ノード間の共通の変数をエッジにとる頂点制約ネットワークでCSPを表現する．エッジで結ばれた2つの頂点を共通変数に関して整合性を保ちながら併合していく．図3.1に併合法の解を見つけるまでの流れを示す．図のCSPでは，変数は X_1, X_2, X_3，変域は $D_1 = \{a, b\}, D_2 = \{e, f\}, D_3 = \{c, d, g\}$，可能な値の組合せとして与えられる制約は $C_1 = \{(X_1 = a, X_2 = f), (X_1 = b, X_2 = e), (X_1 = b, X_2 = f)\}, C_2 = \{(X_2 = e, X_3 = d), (X_2 = f, X_3 = g)\}, C_3 = \{(X_1 = a, X_3 = d), (X_1 = b, X_3 = c), (X_1 = b, X_3 = g)\}$ となる．併合をしていき最終的に1つのノードになったときの変数の可能な値の組合せが解である．併合法の問題点は共通変数に関して併合する際に生成する中間解が非常に大きくなる可能性があることである．

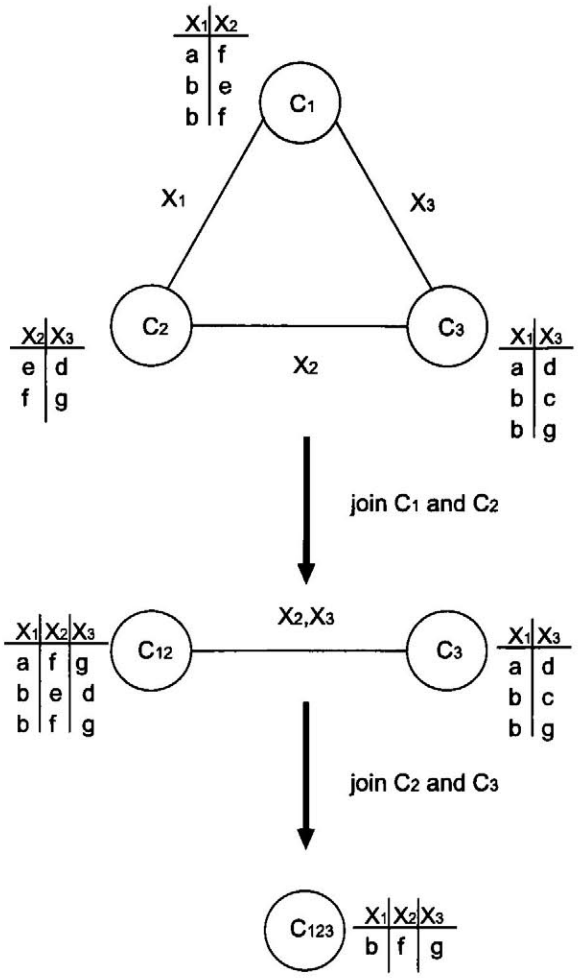

図 3.1　併合法による解探索

3.2　制約充足問題の不完全解法

CSP の不完全解法として代表的に挙げられるのが局所探索法である．多くの局所探索法では，現在の状態から遷移できる近傍についてある評価関数により評価を行い最良の状態へと遷移することにより解を探索する．多くの局所探索法で広く使われているのが次式のような評価関数 $F(\boldsymbol{x})$ である．

$$
\begin{aligned}
F(\boldsymbol{x}) &= \sum_{i=1}^{m} C_i(\boldsymbol{x}), \\
C_i(\boldsymbol{x}) &= \begin{cases} 0 & \text{if } \boldsymbol{x} \text{ satisfies } C_i, \\ 1 & \text{otherwise.} \end{cases}
\end{aligned}
\tag{3.1}
$$

与えられた制約が全て充足するならば，$F(\boldsymbol{x}) = 0$ となる．よって，局所探索法は $F(\boldsymbol{x})$ が小さくなるような近傍に遷移しながら解を探索する．

3.3 MCHC

Mintonらは CSP を解くアルゴリズムとして局所探索法の一種である MCHC(Min-Conflict Hill-Climbing)[11] を提案した．MCHC は次のようなシンプルなアルゴリズムである．

（1）各変数にランダムに値を割り当てる．

（2）制約違反をしている制約に現れる変数の中で式（3.1）の $F(x)$ を最も小さくするような値の入れ替えを各変域の中から選び，その値に更新する．

（3）$F(x) = 0$ なら探索終了．そうでないなら決められた回数だけ (2) を繰り返す．

（4）決められた回数だけ（1）~（3）を繰り返す．

上記のアルゴリズムでは制約違反している制約に現れる全ての変数において近傍 (変域 D_i から変数 X_i のとる値を入れ替えることにより作成する) の評価を行っているが，それ以外にも制約違反している制約に現れる変数の中からランダムに 1 つの変数を選び，その変数において近傍の中から最良な点を選び，その値に更新するという手法も提案されている．MCHC は Selman らが提案した有名な SAT の不完全解法である GSAT[26] によく似たアルゴリズムである．GSAT の場合は，変数のとる値が 0，1 のブール値であるため，近傍は一つの変数を選びその値をフリップ (0 から 1 または 1 から 0 に入れ替える) してやることにより作成する．しかしこれらのアルゴリズムは単純な山登り型アルゴリズムであるため，図 3.2 で示されるような局所最適解（解ではないが近傍によりよい状態がないため変数を更新できない状態）に陥る．図 3.2 において縦軸のエネルギーは式（3.1）によって計算される．そのため，初期値を再割り当てすることが必要である．局所解の問題を克服するために様々なアルゴリズムが提案されている．局所解に陥らなようにするために近傍遷移に確率的な要素を組み込んだ WalkSAT[27] や，局所解に陥ったならそのときの制約の重要度を上げてやり目的関数を作りなおすことにより局所解からの脱出を測る手法[28],[29] といったアルゴリズムが提案されている．

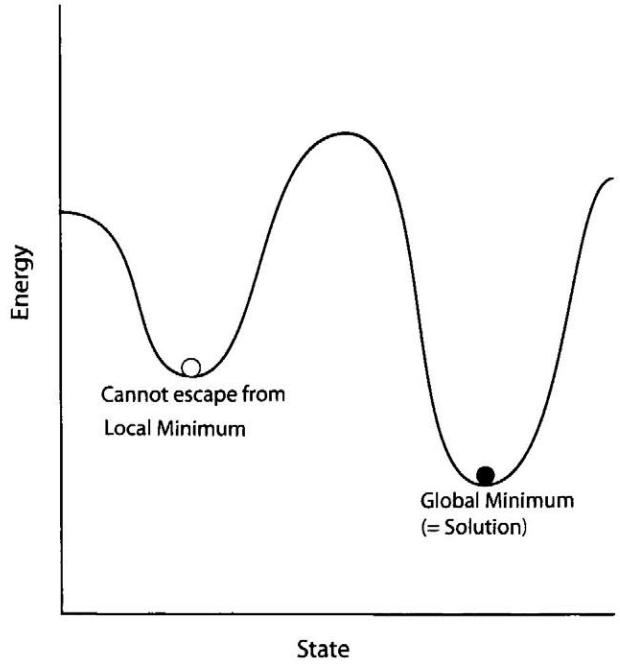

図 3.2　局所解に陥った状態

3.4 GENET

Wang らが提案した離散的なニューラルネットワークである GENET [12], [13] は CSP を高速に解く不完全解法として知られている．これは制約違反を最小にするように各ニューロンは働き，局所解に陥ったならば現在違反している制約の重要度をあげてやることにより局所解からの脱出を図る手法である．またネットワーク構造を拡張してやることによりより一般的な制約も扱うことができる [13].

3.4.1 ネットワーク構造

GENET では，CSP の各変数をクラスタとしてみなす．クラスタ中の各ニューロンは変域の中の各値を示している．ニューロン間のエッジは問題によって与えられた 2 項制約を表している．この 2 項制約は同じエッジによって結合しているニューロンの 2 つの出力はともに 1 になることを禁止するものである．ニューロン間の結合係数は初期値として −1 を与える．

GENETのネットワーク構造例を示すために，以下のような変数，変域，制約の組を考える．

変数 ： X_1, X_2, X_3

変域 ： $D_1 = D_2 = D_3 = \{1, 2, 3\}$

制約 ： X_1のとる値はX_2のとる値より小さくなければならない

X_2とX_3の値は等しくなければいけない

上記の問題を制約ネットワーク図で表現したものを図3.3に示す．図において変数はノード，制約はノード間のエッジによって表現している．与えられた制約は2項関係によって示している．図3.3で例示したCSPをGENETで表現したものを図3.4に示す．

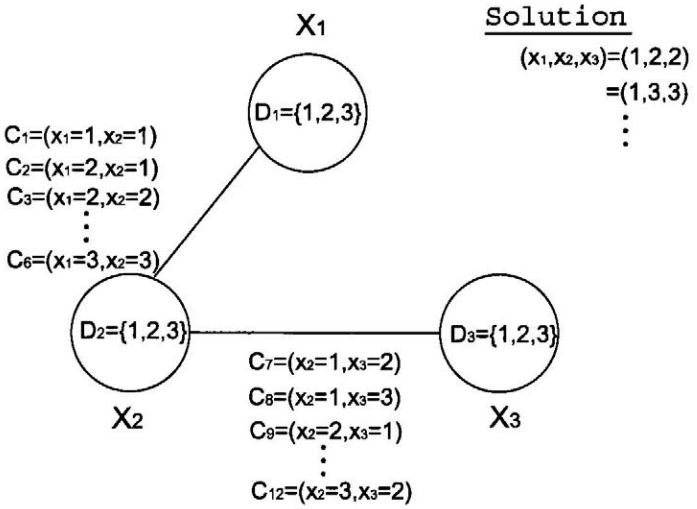

図3.3　制約ネットワーク

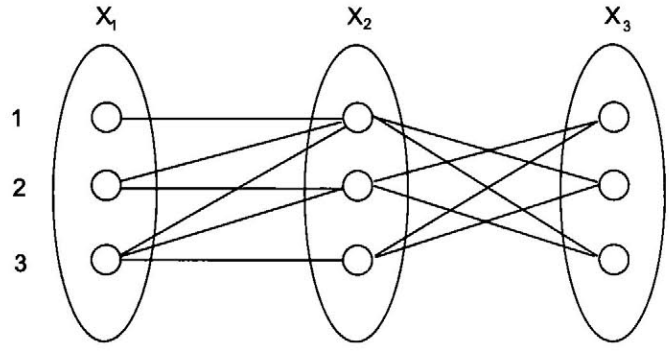

図3.4　GENETのネットワーク構造

変数と変域の値のペア $\langle i, j \rangle$ に用意されたニューロンへの入力は次式によって計算される．

$$I_{\langle i,j \rangle} = \sum_{k \in Z, l \in D_k} W_{\langle i,j \rangle} V_{\langle k,l \rangle}. \tag{3.2}$$

Z は変数の集合，D_k は変数 k のとりうる変域，$W_{\langle i,j \rangle}$ はニューロン $\langle i, j \rangle$ と $\langle k, l \rangle$ の間の結合係数を示している．$V_{\langle k,k \rangle}$ はニューロン $\langle k, l \rangle$ の出力を表しており，$V_{\langle k,k \rangle} = 1$ ならばペア $\langle i, j \rangle$ が成立し，$V_{\langle k,k \rangle} = 0$ ならば成立しないことを意味する．もし，CSP の解がみつかったのなら全てのニューロンの入力は 0 となる．

3.4.2 アルゴリズム

GENET のアルゴリズムは次のようになる．

（1）各変数からランダムに 1 つのニューロンを選んで発火させる (出力を 1 にする).

（2）各変数のなかで最も入力の大きいニューロンを発火する．

（3）全ての変数にたいして（2）を行う．もし，全てのニューロンの値が更新されなかったら次を実行する．

(a) 全ての発火している (出力が 1 になっている) ニューロンの入力が 0 なら解が見つかったので探索終了

(b) その他の場合は，局所解に陥っている学習を行う．

（4）（3）を繰り返す．

3.4.3 学習

GENET は局所解に陥る可能性がある．これは CSP の解は見つけていないが探索空間において現在の状態から改良することができない状態である．このため GENET では局所解からの脱出を図るために次式によりニューロン間の結合係数の学習を行う．

$$W^{t+1}_{\langle i,j \rangle \langle k,l \rangle} = W^{t}_{\langle i,j \rangle \langle k,l \rangle} - V_{\langle k,l \rangle}. \tag{3.3}$$

$W^{t}_{\langle i,j \rangle \langle k,l \rangle}$ は時刻 t におけるニューロン $\langle i, j \rangle$ と $\langle k, l \rangle$ の間の結合係数を表している．結合係数の学習を行うことにより現在制約違反している重みの重要度があがり，局所解から脱出する

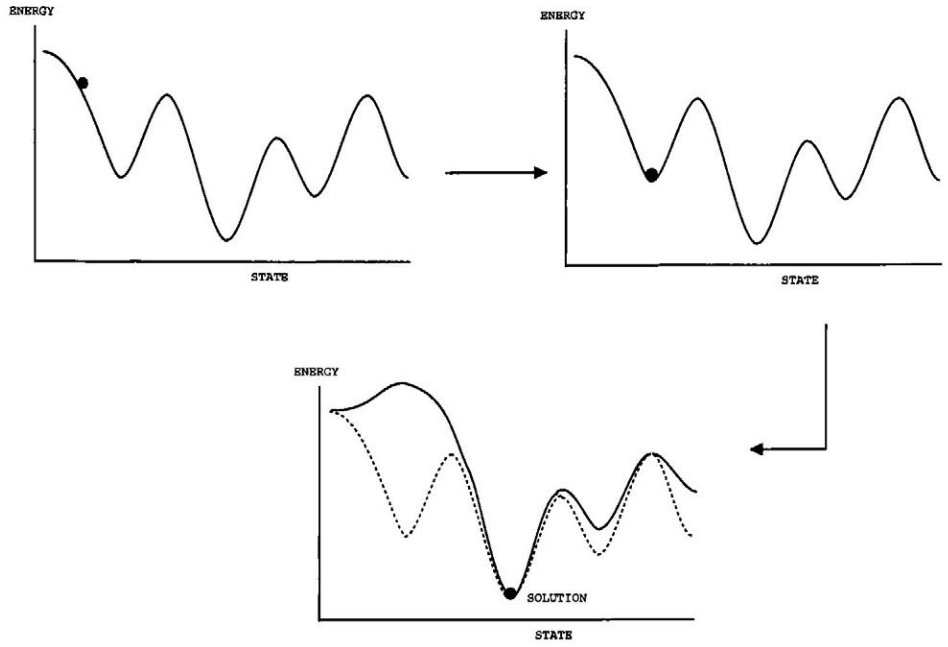

図 3.5　GENET のエネルギー変化

ことができる．局所解から脱出する際のネットワークのエネルギー (全てのニューロンの入力の総和) の変化を図 3.5 に示す．

3.4.4　より一般的な制約を扱う GENET

いままで述べてきた GENET は 2 項制約しか扱うことができない．より一般的な制約を扱うために制約ニューロンの導入を行う [13]．区別のために前述に示したニューロンを VVP(Variable-Value Pair) ニューロンと呼び直す．そして一般的な制約を扱うために新たに制約ニューロンと呼ぶものを導入する．制約ニューロンはその制約に現れている全ての VVP ニューロンと結合している．図 3.4 の GENET に制約ニューロン G を導入したものを図 3.6 に示す．制約 G には変数と値のペア $\langle X_1, 1\rangle, \langle X_2, 1\rangle, \langle X_3, 1\rangle$ が現れているものとする．

制約ニューロン c の入力は次式によって計算される．

$$I_c = \sum_{\langle i,j\rangle \in L} V\langle i, j\rangle. \tag{3.4}$$

L は制約ニューロン c と結合している VVP ニューロンの集合である．VVP ニューロン $\langle i, j\rangle$

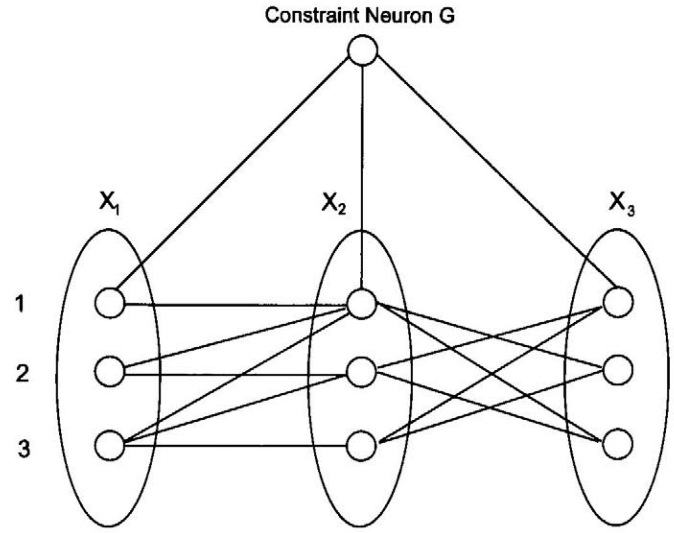

図 3.6　一般的な制約を扱う GENET

の入力は次式によって求められる．

$$I_{\langle i,j\rangle} = \sum_{k\in z, l\in D_k} W_{\langle i,j\rangle\langle k,l\rangle} V_{\langle k,l\rangle} + \sum_{c\in C} W_{c,\langle i,j\rangle} V_{c,\langle i,j\rangle}. \tag{3.5}$$

$V_{c,\langle i,j\rangle}$ は制約ニューロン c の出力を意味し，$W_{c,\langle i,j\rangle}$ は VVP ニューロン $\langle i,j\rangle$ と制約ニューロン c との結合係数を表している．従来の GENET と同様に全ての結合係数の初期値は -1 である．重みの更新式は

$$W^{t+1}_{c,\langle i,j\rangle} = \begin{cases} W^{t}_{c,\langle i,j\rangle} - 1 & \text{if } S_c > 0, \\ W^{t}_{c,\langle i,j\rangle} & \text{otherwise,} \end{cases} \tag{3.6}$$

によって求められる．S_c は制約ニューロン c の状態を表している．次に，GENET が扱う制約について述べる．

3.4.4.1　Illegal 制約

Illegal 制約とは組合せ $L = (\langle x_1, v_1\rangle, \cdots, \langle x_k, v_k\rangle)$ は禁止するというような多項制約を意味する．つまり $illegal(\langle x_1, v_1\rangle, \cdots, \langle x_k, v_k\rangle)$ が与えられたならば，$\langle x_1, v_1\rangle, \cdots, \langle x_k, v_k\rangle$ のペアが全て成立することは許されない．Illegal 制約ニューロン ill の状態は次式によって求める．

$$S_{ill} = I_{ill} - (k-1). \tag{3.7}$$

状態 S_{ill} は Illegal 制約ニューロン ill に結合している VVP ニューロンに関して，出力が 1 になっている VVP のニューロンの数が $k-1$ 個未満になれば負になる．この場合，0 を出力し

ていたVVPニューロンのうちでどれか1つが発火しても制約を違反することはない．この状態の場合，Illegal制約ニューロン ill の出力は0となる．$S_{ill} = 0$ の場合は，0を出力していたVVPニューロンのうちでどれか1つが発火すると制約を違反してしまう．この場合は制約ニューロンは1を出力しているVVPニューロンには0を出力し，0を出力しているVVPニューロンには1を出力する．もし $S_{ill} > 0$ ならば制約を違反しているので，結合している全てのVVPニューロンに1を出力する．

制約ニューロン ill からVVPニューロン $\langle i, j \rangle$ への出力は次式によって計算される．

$$V_{ill,\langle i,j \rangle} = \begin{cases} 0 & \text{if } S_{ill} < 0, \\ 1 + S_{ill} - V_{\langle i,j \rangle} & \text{otherwise.} \end{cases} \tag{3.8}$$

3.4.4.2 Atmost制約

Atmost制約とは $atmonst(N, \langle x_1, v_1 \rangle, \cdots, \langle x_k, v_k \rangle)$ が与えられたならば，$\langle x_1, v_1 \rangle, \cdots, \langle x_k, v_k \rangle$ のなかで多くとも N 個のペアは成り立つことを意味する．Atmost制約ニューロン atm の状態 S_{atm} は次式によって計算される．

$$S_{atm} = I_{atm} - N. \tag{3.9}$$

Atmost制約ニューロン atm からVVPニューロン $\langle i, j \rangle$ の出力は

$$V_{atm,\langle i,j \rangle} = \begin{cases} 0 & \text{if } S_{ill} < 0, \\ 1 - \mathrm{Max}\{V_{\langle i,k \rangle} | k \in Val\} & \text{if } S_{atm} = 0, \\ 1 & \text{otherwise,} \end{cases} \tag{3.10}$$

となる．S_{atm} はAtmost制約ニューロン atm に結合しているVVPニューロンの発火総数が N 未満なら負になる．この場合，Atmost制約ニューロンは0を出力する．結合しているVVPニューロンの発火総数が N の場合に，Atmost制約に結合しているVVPニューロンの中で変数 i のクラスターに属する $V_{\langle i,k \rangle}, \cdots, V_{\langle i,l \rangle}$ のうちどれかが発火しているならばVVPニューロンは0を出力し，すべて発火していないならばVVPニューロンは1を出力する．結合しているVVPニューロンの発火総数が $N+1$ 以上なら結合している全てのVVPニューロンに対してAtmost制約ニューロンは1を出力する．

3.5 連続値変数を用いる解法

これまでに紹介した解法は変数は離散値をとる解法であり，図3.7で示すように，離散的な探索空間で探索を進めてゆく手法であった．これに対し，離散的な問題をいったん連続的

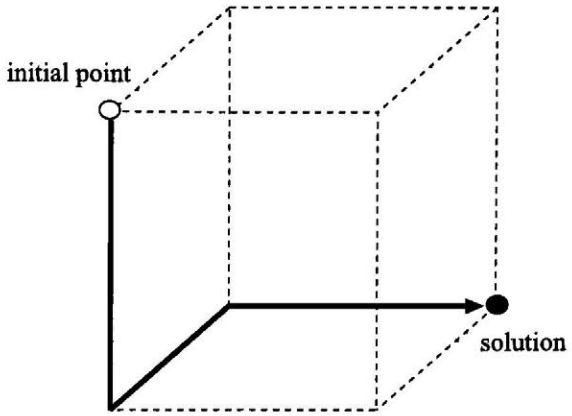

図 3.7　離散値変数を用いる解法

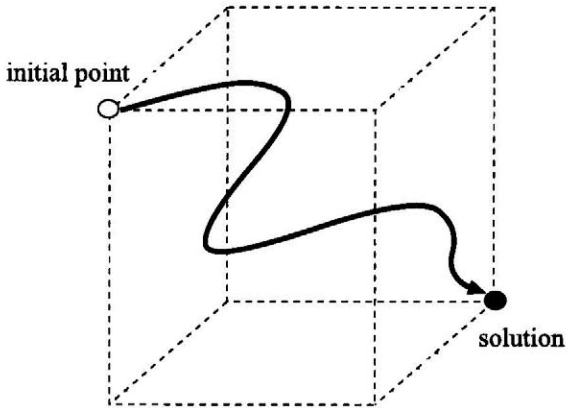

図 3.8　連続値変数を用いる解法

な問題に緩和し，連続的な探索空間の中で探索を進めていく手法が提案されている．もし，連続値をとり，全ての変数が同時のタイミングで更新できる効率的な解法が実現できるのならば，並列化での活用，ハードウェアでの実装でのスピードアップなどを考慮にいれると大いに意義のある課題であるといえる．

Gu らは，離散的な問題である SAT を次のように定義される連続値の問題 UniSAT に変換することを提案した[15],[16]．UniSAT では離散的な探索空間 $\boldsymbol{x} \in \{0,1\}^n$ を実数空間 $\boldsymbol{y} \in E^n$ に拡張する．

(UniSAT)　find $\boldsymbol{y}$ such that

$$\boldsymbol{y} \text{ minimize } f(\boldsymbol{y}),$$
$$\text{where} f(\boldsymbol{y}) = \sum_{i=1}^{m} c_i(\boldsymbol{y}). \tag{3.11}$$

節関数 $c_i(\boldsymbol{x})$ はリテラル関数 $q_{i,j}(y_j)$ $(i \le j \le n)$ の積によって計算される．

$$c_i = \prod_{j=1}^{n} q_{i,j}(y_j). \tag{3.12}$$

リテラル関数 $q_{i,j}(y_j)$ は y_j の値がリテラル x_j を真とするかを判定する関数であり，UniSAT5 と UniSAT7 という２つのモデルが提案されている．$q_{i,j}(y_j)$ は UniSAT5 では

$$q_{i,j}(y_j) = \begin{cases} |y_j - 1| & \text{if literal } x_j \text{ is in clause } C_i, \\ |y_j + 1| & \text{if literal } \hat{x}_j \text{ is in clause } C_i, \\ 1 & \text{if neither } x_j \text{ nor } \hat{x}_j \text{ is in clause } C_i, \end{cases} \tag{3.13}$$

となり，UniSAT7 ではリテラル関数は次のように定義される．

$$q_{i,j}(y_j) = \begin{cases} (y_j - 1)^2 & \text{if literal } x_j \text{ is in clause } C_i, \\ (y_j + 1)^2 & \text{if literal } \hat{x}_j \text{ is in clause } C_i, \\ 1 & \text{if neither } x_j \text{ nor } \hat{x}_j \text{ is in clause } C_i. \end{cases} \tag{3.14}$$

最後に $\boldsymbol{x}$ と $\boldsymbol{y}$ は次式によって対応付けられる．

$$x_i = \begin{cases} 1 & \text{if } y_i = 1\,, \\ 0 & \text{if } y_i = -1, \\ undefined & \text{otherwise.} \end{cases} \tag{3.15}$$

Gu らはこの UniSAT に対して幾つかのアルゴリズムを提案している．UniSAT を解く最も基礎的なアルゴリズムである SAT6.0 は次のような手順で解を探索する．

（1）$\boldsymbol{y}$ に初期値を割り当てる．

（2）幾つかの y_i に関して $f(\boldsymbol{y})$ を最小化できるなら，それを実現し，$f(\boldsymbol{y})$ を更新する．

（3）もし $\boldsymbol{y}$ が解に十分近いなら，$\boldsymbol{y}$ を丸め，それを $\boldsymbol{x}$ とする．

（4）もし局所解に陥ったならば脱出を測る．

（5）解がみつかるまで（2）～（4）を繰り返す．

上記アルゴリズムのステップ（2）において目的関数を最小化するための手法としては勾配法やニュートン法をはじめとする最適化手法を用いることができる．また，ステップ（4）で局所解から脱出する方法の 1 つとしては，最大 n 変数の真値を否定する手法が挙げられる．Gu は上記のアルゴリズム以外にも UniSAT に対する幾つかの連続値の解法を提案しているが，実験により Gu が提案した連続値の手法は離散値の解法と比べ，とても時間がかかってしまうため，大きな SAT 問題を解くのには適していないと結論付けている．また Kwok ら

は連続値をとる解法である SONN (Self-Organizing Neural Network)[30] を提案し，N-Queen 問題に適用している．しかし，この論文は SONN の最適化能力を明らかにすることのみに焦点をあてており，SONN の効率性は考慮しておらず，実際に N-Queen 問題の実験結果は小さなサイズの N-Queen 問題に対してもよい結果は得られていない．このように離散的な問題である SAT や CSP に対しては一般的に，離散値をとる解法に比べ，連続値をとる解法は解を得るまでに時間がかかってしまうとされていた．しかし，永松らは SAT に対して連続値をとる解法である LPPH[2]~[5] と呼ばれる手法を提案し，実験結果より，従来の離散値をとる GSAT などの SAT の解法に比べ効率的に解を探索することを示した．この解法もまず SAT をそれと等価な連続的な問題に変換し，その問題に LPPH を適用することにより SAT の解を探索する．次章に LPPH の定義とその性質などを述べる．

第4章　ニューラルネットワーク LPPH

永松らは SAT を解く LPPH と呼ばれるニューラルネットワークを提案した．まずこの手法では，離散的な問題である SAT をそれと等価な連続的な問題 CONSAT に変換する．この CONSAT に対して LPPH を適用することにより SAT の解を求める．

4.1　Continuous Valued Satisfiability Problem

連続値をとる充足可能性問題である CONSAT（Continuous Valued Satisfiability Problem）および関数 $g_{ir} : [0,1]^n \to [0,1]$，$h_r : [0,1]^n \to [0,1]$ を定義する．

(CONSAT)　find $\boldsymbol{x}$ such that

$$h_r(\boldsymbol{x}) = 0, \quad \forall r = 1, 2, \cdots, m,$$

$$\boldsymbol{x} \in [0,1]^n, \tag{4.1}$$

$$h_r(\boldsymbol{x}) = \prod_{i=1}^{n} g_{ir}(\boldsymbol{x}), \tag{4.2}$$

$$g_{ir}(\boldsymbol{x}) = \begin{cases} x_i & \text{if } \bar{x}_i \text{ appears in } C_r, \\ 1 - x_i & \text{if } x_i \text{ appears in } C_r, \\ 1 & \text{otherwise.} \end{cases} \tag{4.3}$$

SAT における変数の取る値は 0，1 の離散値であるが上記の CONSAT の定義では，変数は 0 から 1 の間の連続値を値としてとる．$h_r(\boldsymbol{x})$ はクローズ C_r の充足のしていない度合いを表し，これを非充足度関数と呼ぶ．もし C_r が充足しているならば，$h_r(\boldsymbol{x}) = 0$ となり，それ以外ならば $h_r(\boldsymbol{x}) > 0$ となる．CONSAT の解は，幾つかの変数は 0 から 1 の間の非整数値をとってしまう可能性がある．これらの変数を 0 または 1 の値のどちらでも良いという“ don't care” 変数としてみなすことにより，SAT と CONSAT は等価なものとなる．

4.2 LPPHの力学系

LPPHは次式によって定義されるラグランジュ関数をもつラグランジュ法に基づいている.

$$F(\boldsymbol{x}, \boldsymbol{w}) = \sum_{r=1}^{m} w_r h_r(\boldsymbol{x}),$$
$$\text{where } \boldsymbol{x} \in [0,1]^n,\ \boldsymbol{w} \in (0,\infty)^m. \tag{4.4}$$

上記の定義より，$\boldsymbol{x}$がCONSATの解であるその時に限り，$F(\boldsymbol{x}, \boldsymbol{w}) = 0$であることは明らかである.

LPPHの力学系の定義は次のようになる.

$$\frac{dx_i}{dt} = -x_i(1-x_i)\frac{\partial F(\boldsymbol{x}, \boldsymbol{w})}{\partial x_i}, \quad i = 1, 2, \cdots, n, \tag{4.5}$$

$$\frac{dw_r}{dt} = \frac{\partial F(\boldsymbol{x}, \boldsymbol{w})}{\partial w_r} - \alpha w_r$$
$$= h_r(\boldsymbol{x}) - \alpha w_r, \quad r = 1, 2, \cdots, m. \tag{4.6}$$

wはクローズの重みであり，式（4.6）にあるように，クローズの充足の度合いにより増減をする．αはLPPHの効率性に影響を与える[3]パラメータであり，これを減衰係数と呼ぶ．式（4.5），（4.6）の連立方程式を解くことによりSATの解を得ることができる．また，$\alpha = 0$の場合，LPPHは次のような性質を持つことが証明されている[2].

- LPPHの全ての平衡点はSATの解であり，その逆も真である.
- LPPHはCONSATの解に近づいたとき，その解に収束する.

4.3 $h_r(\boldsymbol{x})$の新たな定義

式(4.2)はクローズがどのくらい充足をしていないかを計算する式であったが，この定義は唯一のものではない．例えば次のようなクローズC_1を考えてみよう.

$$C_1 = x_1 \vee x_2 \vee x_3 \vee x_4.$$

また，CONSATにおけるの2つの変数割当$\boldsymbol{x}_1$，$\boldsymbol{x}_2$は次のような値を取るとする.

$$\boldsymbol{x}_1 = (x_1, x_2, x_3, x_4) = (0.5, 0.5, 0.5, 0.5),$$

$$\boldsymbol{x}_2 = (x_1, x_2, x_3, x_4) = (0.1, 0.9, 0.85, 0.9).$$

表 4.1　SAT problems

name	n	m	max	
rg1	100	430	3	randomly generated 3-SAT
rg2	200	860	3	
hm1	100	762	9	hamilton circuit
hm2	225	2795	14	
tp1	319	941	10	test pattern generation for circuits
tp2	196	529	10	
q15	225	5195	15	15-Queen problem
q20	400	12560	20	20-Queen problem

この場合，x_1 と x_2 はどちらがより C_1 を充足させるだろうか？C_1 は $x_1 \sim x_4$ のいずれかの変数が 1 となれば充足するので，直観的には，x_2 の方が C_1 をより充足させていると考えるのが自然である．しかし，式（4.2）から，x_1，x_2 による C_1 の非充足度関数は，$h_r(x_1) = 0.0625$，$h_r(x_2) = 0.06885$ となる．つまり，式（4.2）の定義式では x_1 がより C_1 を充足していると計算してしまう．

そこで次のような定義式を用いて非充足度関数 $h_r(x)$ を計算することを提案する [6]．

$$h_r(x) = min(g_{1r}, g_{2r}, \cdots, g_{nr}). \tag{4.7}$$

式(4.7)から，x_1，x_2 による C_1 の非充足度関数は，$h_r(x_1) = 0.5$，$h_r(x_2) = 0.1$ となる．これは前述した直感的な評価と一致する．式（4.2）と（4.7）の違いは $h_r(x)$ の計算に MIN 演算を用いるかまたは乗算を用いるかの違いである．

計算機実験により式（4.2）と式（4.7）により計算される $h_r(x)$ についての解探索の効率性について調べた．表 4.1 に実験に用いた問題を示す．表 4.1 において n, m, max はそれぞれ変数の数，クローズの和，クローズ中のリテラルの最大値を表している．図 4.1 は両方の手法によって得られた結果を示している．実験において LPPH におけるパラメータである減衰係数 α は 0.6 に設定した [3]．図 4.1 において縦軸は探索の初期値から解を見つけるまでに要した CPU 時間の平均を示している．平均は 1 つの問題に対して LPPH の初期値を 30 回

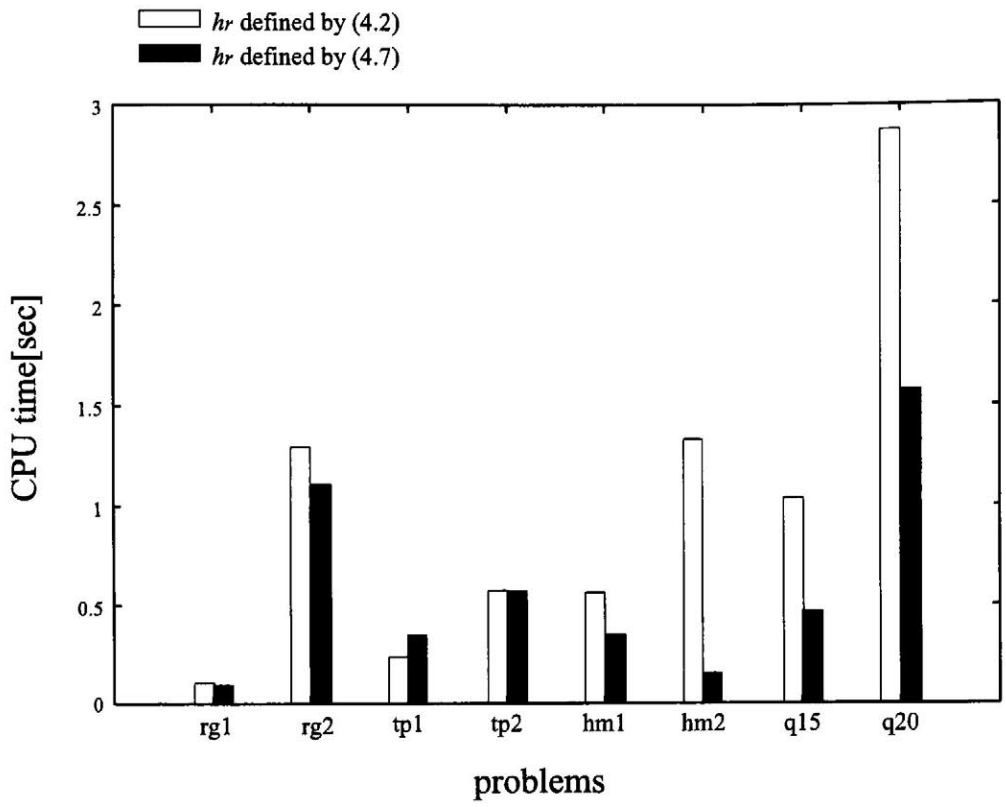

図 4.1　各 h_r の平均 CPU 時間の比較

替え，試行して得られた結果から計算した．結果より，クローズ中のリテラルの数が多い問題に対しては式（4.7）によって $h_r(x)$ を計算した場合の方が良い結果を示しいる．これはクローズ中のリテラルの数が多い場合，$h_r(x)$ を式（4.2）により計算すると前述に示したような直観的に充足のしていない度合いを評価する場合とは異なった評価を下す場合が多くなってしまい，その結果探索の効率性に影響を及ぼしているのではないかと考えられる．

SAT における 1 つのクローズは“ クローズ中の少なくとも 1 つのリテラルが真である”という制約として捉えることができる．次の節では，式（4.7）と同じアプローチに基づいて，SAT の制約に比べより一般的な CSP の制約に対して，非充足度を計算する方法を提案する．

第5章　制約充足問題を解くニューラルネットワーク LPPH-CSP

5.1　LPPH-CSPの力学系

CSPを解くために4章で示した力学系を拡張する．この力学系をLPPH-CSP[6],[8]と呼ぶ．ニューラルネットワークの変数x_{ij}は“CSPの変数X_iが変域D_iのj番目の値をとる”ことの度合いを表す変数とするとLPPH-CSPの力学系は次のようになる．

$$\frac{dx_{ij}}{dt} = x_{ij}(1-x_{ij})\sum_{r=1}^{m} w_r s_{rij}(\boldsymbol{x}), \text{ for all VVP } x_{ij}, \tag{5.1}$$

$$\frac{dw_r}{dt} = h_r(\boldsymbol{x}) - \alpha w_r,\ r = 1,2,\cdots,m. \tag{5.2}$$

式（5.2）においてw_rは制約C_rの重みを表しており，C_rの非充足度関数$h_r(\boldsymbol{x})$の値に基づいて増減する．$h_r(\boldsymbol{x})$は式（4.2），（4.7）と同様にC_rが満されるならば0，それ以外ならば正の値を返す関数をである．また，関数$s_{rij}(\boldsymbol{x})$はC_rを充足させるためにx_{ij}に与える力を表し，これを充足指示関数と呼ぶ．つまり，x_{ij}はその変数が現れている全ての制約を充足するように値を変化させる．w_rはC_rが充足しているならば減衰係数αにw_rを掛けた分だけ小さくなり，C_rが充足していないならば$h_r(\boldsymbol{x}) - \alpha w_r$により値を変化する．SATの節$C_r$は“$C_r$中のリテラル$l_j$の何れかが真となる”という制約として捉えることができる．その場合，LPPHにおいてこの制約を充足するために変数x_iに与える力は式（4.4），（4.5）より$s_{ri}(\boldsymbol{x}) = -\partial h_r(\boldsymbol{x})/\partial x_i$となる．CSPに対しては，2.2節で示した制約に対してそれぞれ$h_r(\boldsymbol{x})$, $s_{rij}(\boldsymbol{x})$を定義すれば，これらの関数にしたがってLPPH-CSPは制約充足をおこなうことができる．

図5.1にLPPH-CSPのニューラルネットワークイメージを示す．変数ニューロンは対応するその変数が現れている制約ニューロンと結合している．例えば図5.1において，変数ニューロンx_{11}はx_{11}が現れれる制約ニューロンC_1，C_2と結合しており，その制約ニューロンにx_{11}の値を出力する．各制約ニューロンはその制約に現れている変数ニューロンから値を受け取り，その値からそれぞれの変数に送る充足指示関数を計算し，変数ニューロンに出力す

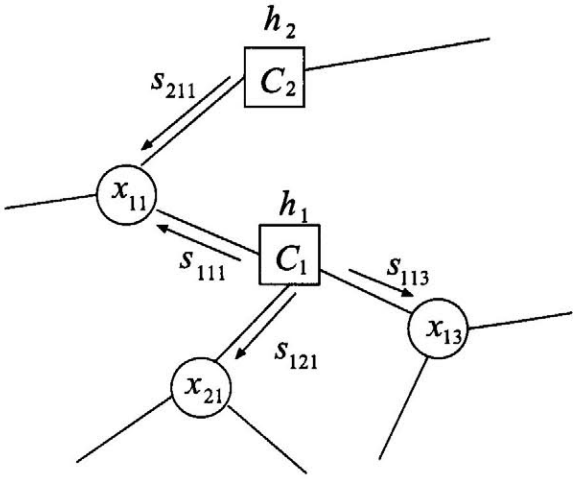

図 5.1　LPPH-CSP のニューラルネットワークイメージ

る．各変数ニューロンはその変数が出現している全ての制約ニューロンから受けとる充足指示関数の値から自分がどう変化するかを決定し，自分の値を更新する．ニューラルネットワークは全ての制約ニューロンの非充足度関数が 0 になったならば，収束し，そのときの変数ニューロンの値が CSP の解となる．

LPPH-CSP において $h_r(\boldsymbol{x})$，$s_{rij}(\boldsymbol{x})$ をどう定義するかが重要な問題となる．次節では $h_(\boldsymbol{x})$，$s_{rij}(\boldsymbol{x})$ について 3 つの手法を定義し，それらの手法の効率性について議論する．

5.1.1　LPPH-CSP(1)

節 4.2 で述べたように充足可能性問題を解く従来の LPPH では乗算により，CNF 論理式の節に対して現在の変数割当 $\boldsymbol{x}$ の非充足度関数 $h_r(\boldsymbol{x})$ を計算し，各変数は $wh_r(\boldsymbol{x})$ の総和をその変数で偏微分した量に基づいて値を変化させていく．

LPPH-CSP(1) では，この考え方を制約充足問題における制約の $h_r(\boldsymbol{x})$，$s_r(\boldsymbol{x})$ の定義式に応用する．つまり $s_{rij}(\boldsymbol{x})$ は次式のように $h_r(\boldsymbol{x})$ を $\boldsymbol{x}$ について偏微分することにより計算される．

$$s_{rij}(\boldsymbol{x}) = -\frac{\partial h_r(\boldsymbol{x})}{\partial x_{ij}}. \tag{5.3}$$

次に各 ALT$(1,n)$，ALF$(1,n)$，AMT$(1,n)$，AMF$(1,n)$ の $h_r(\boldsymbol{x})$，$s_{rij}(\boldsymbol{x})$ は次のように定義される．

- C_r=ALT$(1, S)$

$$h_r(\boldsymbol{x}) = \prod_{x_{ij} \in S} (1 - x_{ij}). \tag{5.4}$$

よって，式（5.3）より $s_{rij}(\boldsymbol{x})$ は次式のようになる．

$$s_{rij}(\boldsymbol{x}) = \prod_{\substack{x_{kl} \in S \\ (k,l) \neq (i,j)}} (1 - x_{kl}). \tag{5.5}$$

- C_r=ALF$(1, S)$

$$h_r(\boldsymbol{x}) = \prod_{x_{ij} \in S} x_{ij}, \tag{5.6}$$

$$s_{rij}(\boldsymbol{x}) = - \prod_{\substack{x_{kl} \in S \\ (k,l) \neq (i,j)}} x_{kl}. \tag{5.7}$$

- C_r=AMT$(1, S)$

$$h_r(\boldsymbol{x}) = \frac{\displaystyle\sum_{x_{ij} \in S} \sum_{\substack{x_{kl} \in S \\ (k,l) \neq (i,j)}} x_{ij} x_{kl}}{2_{n_r}C_2}, \tag{5.8}$$

$$s_{rij}(\boldsymbol{x}) = -\frac{\displaystyle\sum_{\substack{x_{kl} \in S \\ (k,l) \neq (i,j)}} x_{kl}}{{}_{n_r}C_2}. \tag{5.9}$$

- C_r=AMF$(1, S)$

$$h_r(\boldsymbol{x}) = \frac{\displaystyle\sum_{x_{ij} \in S} \sum_{\substack{x_{kl} \in S \\ (k,l) \neq (i,j)}} (1 - x_{ij})(1 - x_{kl})}{2_{n_r}C_2}, \tag{5.10}$$

$$s_{rij}(\boldsymbol{x}) = \frac{\displaystyle\sum_{\substack{x_{kl} \in S \\ (k,l) \neq (i,j)}} (1 - x_{kl})}{{}_{n_r}C_2}. \tag{5.11}$$

上記の定義では，制約 ALT(n, S), ALF(n, S), AMT(n, S), AMF(n, S) において $n = 1$ の場合しか説明していない．しかし，$n > 1$ の場合も同様にして $h_r(\boldsymbol{x})$, $s_{rij}(\boldsymbol{x})$ を定義することができる．しかし，これらの場合では関数における項の数が非常に大きくなってしまう．例えば，ALT(n, S) に対する $h_r(\boldsymbol{x})$ の項は ${}_{|S|}C_{|S|-n-1}$ となる．

5.1.2 LPPH-CSP(2)

4.3節ではファジィ論理におけるMIN演算を用いて充足可能性問題の節の $h_r(\boldsymbol{x})$ を計算し，従来の乗算での $h_r(\boldsymbol{x})$ よりも効率的に解を探索することが可能になることを示した．5.1.2節では，この考え方を一般化することにより制約充足問題の制約の $h(\boldsymbol{x})$ を計算する．

- C_r=ALT(n, S)

$$h_r(\boldsymbol{x}) = 1 - \mathrm{NMax}(n, S), \tag{5.12}$$

$$s_{rij}(\boldsymbol{x}) = \begin{cases} 1 - \mathrm{NMax}(n+1, S) & \text{if } x_{ij} \geq \mathrm{NMax}(n, S), \\ 1 - \mathrm{NMax}(n, S) & \text{otherwise,} \end{cases}$$

where NMax(n, S) = nth maximum value in S. (5.13)

- C_r=ALF(n, S)

$$h_r(\boldsymbol{x}) = \mathrm{NMin}(n, S) \tag{5.14}$$

$$s_{rij}(\boldsymbol{x}) = \begin{cases} -\mathrm{NMin}(n+1, S) & \text{if } x_{ij} \leq \mathrm{NMin}(n, S), \\ -\mathrm{NMin}(n, S) & \text{otherwise,} \end{cases}$$

where NMin(n, S) = nth minimum value in S. (5.15)

- C_r=AMT(n, S)

$$h_r(\boldsymbol{x}) = \mathrm{NMax}(n+1, S), \tag{5.16}$$

$$s_{rij}(\boldsymbol{x}) = \begin{cases} -\mathrm{NMax}(n, S) & \text{if } x_{ij} \leq \mathrm{NMax}(n+1, S), \\ -\mathrm{NMax}(n+1, S) & \text{otherwise.} \end{cases} \tag{5.17}$$

- C_r=AMF(n, S)

$$h_r(\boldsymbol{x}) = 1 - \mathrm{NMin}(n+1, S), \tag{5.18}$$

$$s_{rij}(\boldsymbol{x}) = \begin{cases} 1 - \mathrm{NMin}(n, S) & \text{if } x_{ij} \geq \mathrm{NMin}(n+1, S), \\ 1 - \mathrm{NMin}(n+1, S) & \text{otherwise.} \end{cases} \tag{5.19}$$

例として次のような制約，変数割当を考えてみよう．

$C_r = \mathrm{ALT}(2, S = \{x_{11}, x_{21}, x_{31}, x_{41}\})$,

where $x_{11} = 0.9, x_{21} = 0.8, x_{31} = 0.7$ and $x_{41} = 0.6$.

この例の場合，LPPH-CSP(2) では，$h_r(\boldsymbol{x})$，$s_{rij}(\boldsymbol{x})$ を次のように計算する．

$$
\begin{aligned}
h_r(\boldsymbol{x}) &= 1 - \mathrm{NMax}(2, S) = 0.2, \\
s_{r11}(\boldsymbol{x}) &= s_{r21}(\boldsymbol{x}) = 1 - \mathrm{NMax}(3, S) = 0.3, \\
s_{r31}(\boldsymbol{x}) &= s_{r41}(\boldsymbol{x}) = 1 - \mathrm{NMax}(2, S) = 0.2.
\end{aligned}
$$

5.1.3 LPPH-CSP(3)

LPPH-CSP(1) では制約中の変数を乗算することにより，$h_r(\boldsymbol{x})$ を求め，その $h_r(\boldsymbol{x})$ を偏微分することにより $s_{rij}(\boldsymbol{x})$ を計算する．一方，LPPH-CSP(2) では NMin，NMax 関数を用いることにより $h_r(\boldsymbol{x})$，$s_{rij}(\boldsymbol{x})$ を計算した．つまり，LPPH-CSP(1) は制約中の全ての変数を用いて $h_r(\boldsymbol{x})$ を計算し，その制約を充足するために，各変数は自分以外の全ての変数の値に基づいて自分の値を変化させる．LPPH-CSP(2) は制約を充足しているかどうかを判断することができるある 1 つの変数の値を用いて $h_r(\boldsymbol{x})$ を計算する．また，制約を充足させるための力を計算するために各変数は自分以外のある 1 つの変数の値を用いる．そこで，LPPH-CSP(3) では上記の 2 つの手法の考え方を組み合わせる．各制約の $h_r(\boldsymbol{x})$，$s_{rij}(\boldsymbol{x})$ を求めるために制約に現れる $l+1$ 個の変数の値を用いる．定義式は次のようになる．

- C_r=ALT(n, S)

$$
h_r(\boldsymbol{x}) = \prod_{u=0}^{l} (1 - \mathrm{NMax}(n+u, S)), \tag{5.20}
$$

$$
s_{rij}(\boldsymbol{x}) = \begin{cases} \prod_{u=1}^{l+1} (1 - \mathrm{NMax}(n+u, S)) & \text{if } x_{ij} \geq \mathrm{NMax}(n, S), \\ 1 - \mathrm{NMax}(n, S) & \text{otherwise.} \end{cases} \tag{5.21}
$$

- C_r=ALF(n, S)

$$
h_r(\boldsymbol{x}) = \prod_{u=0}^{l} \mathrm{NMin}(n+u, S), \tag{5.22}
$$

$$
s_{rij}(\boldsymbol{x}) = \begin{cases} -\prod_{u=1}^{l+1} \mathrm{NMin}(n+u, S) & \text{if } x_{ij} \leq \mathrm{NMin}(n, S), \\ -\mathrm{NMin}(n, S) & \text{otherwise.} \end{cases} \tag{5.23}
$$

- C_r=AMT(n, S)

$$h_r(\boldsymbol{x}) = \frac{\sum_{u=1}^{l+1} \mathrm{NMax}(n+u, S)}{l+1}, \tag{5.24}$$

$$s_{rij}(\boldsymbol{x}) = \begin{cases} -\frac{\sum_{u=0}^{l} \mathrm{NMax}(n-u,S)}{l+1} & \text{if } x_{ij} \leq \mathrm{NMax}(n+1, S), \\ -\frac{\sum_{u=1}^{l+1} \mathrm{NMax}(n+u,S)}{l+1} & \text{otherwise.} \end{cases} \tag{5.25}$$

- C_r=AMF(n, S)

$$h_r(\boldsymbol{x}) = \frac{\sum_{u=1}^{l+1} (1 - \mathrm{NMin}(n+u, S))}{l+1}, \tag{5.26}$$

$$s_{rij}(\boldsymbol{x}) = \begin{cases} \frac{\sum_{u=0}^{l} (1-\mathrm{NMin}(n-u,S))}{l+1} & \text{if } x_{ij} \geq \mathrm{NMin}(n+1, S), \\ \frac{\sum_{u=1}^{l+1} (1-\mathrm{NMin}(n+u,S))}{l+1} & \text{otherwise.} \end{cases} \tag{5.27}$$

5.2 実験結果

実験で用いた各手法はC++言語により実装し，それらをPetium IV (2.40GHz)のCPUが搭載されたPCで計算機実験を行った．本節の実験ではN-Queen問題，car sequencing problem [20]，random CSPを問題として用いた．car sequencing problemはCSPLib（http://4c.ucc.ie/ tw/csplib/）に記載されているベンチマーク問題を使用した．random CSPは変数100，各変数の変域数40，制約数1200のランダムに生成した問題と変数160，各変数の変域数40，制約数2000のランダムに生成した問題である．

5.2.1 各ダイナミクスの効率の比較

LPPH-CSP(1)，LPPH-CSP(2), LPPH-CSP(3)のダイナミクスにおける解探索の効率性を実験により調べた．また，実験で用いた問題をSATで表現し，そのSATを従来のLPPH（$h_r(\boldsymbol{x})$の計算には式（4.7）で定義したものを用いる）で解いた場合に要したCPU時間（sec）についても調べた．N-Queen問題を表現する場合，CSPでは節2.2で述べたALT$(1, S)$とAMT$(1, S)$を用いて表現する．SATで表現する場合は，ALT$(1, S)$は1つの節によって表現することができるが，AMT$(1, S)$を表現する場合にはS中の任意の2つのペアについて次式のような節が必要となる．

$$(\bar{x}_{1i} \vee \bar{x}_{1j}) \qquad \text{for} \quad i < j.$$

表5.1はN-Queen問題を解いた場合の各手法が要した平均CPU時間と試行回数の中で打ち切り時間内に解を発見できた回数を示している．平均CPU時間は各手法を初期値を30回変え

300秒の打ち切り時間で試行し，解いた場合の結果から計算した．表5.1からLPPH-CSP(2), LPPH-CSP(3)が他の手法に比べて効率的な解探索をしていることを確認した．

car sequencing problemやrandom CSPは一般的な制約で表現する場合，AMT(n, S)制約が必要となる．n，Sのサイズとしては$n = 80$，$|S| = 200$となるような問題を実験で用いる．SATでこれらの問題を表現する場合には1つのAMT(n, S)制約に対して，${}_{200}C_{81}$個の節を要し，問題のサイズが膨大なものになってしまう．つまり，これらの問題をSATで表現して解くことは，明らかに妥当ではない．

表5.1　N-Queen問題に対するLPPH，LPPH-CSP(1)~ (3)の平均CPU時間

problem	LPPH	LPPH-CSP(1)	LPPH-CSP(2)	LPPH-CSP(3) with $l = 1$	LPPH-CSP(3) with $l = 2$
10-Queen	0.070 s (30/30)	0.0253 s (30/30)	0.003 s (30/30)	0.006 s (30/30)	0.006 s (30/30)
20-Queen	1.593 s (30/30)	2.2550 s (30/30)	0.013 s (30/30)	0.019 s (30/30)	0.020 s (30/30)
30-Queen	13.048 s (30/30)	10.0810 s (30/30)	0.023 s (30/30)	0.048 s (30/30)	0.060 s (30/30)
40-Queen	49.304 s (30/30)	27.2283 s (30/30)	0.073 s (30/30)	0.138 s (30/30)	0.188 s (30/30)
50-Queen	116.169 s (28/30)	79.1430 s (30/30)	0.219 s (30/30)	0.389 s (30/30)	0.555 s (30/30)

表 5.2　N-Queen 問題に対する LPPH-CSP(2)~(3) に対する平均 CPU 時間

problem	LPPH-CSP(2)	LPPH-CSP(3) with $l = 1$	LPPH-CSP(3) with $l = 2$
100-Queen	1.234 s (30/30)	4.066 s (30/30)	9.415 s (30/30)
110-Queen	1.652 s (30/30)	4.738 s (30/30)	11.668 s (30/30)
120-Queen	1.913 s (30/30)	6.736 s (30/30)	14.263 s (30/30)
130-Queen	1.957 s (30/30)	9.188 s (30/30)	23.005 s (30/30)
140-Queen	2.853 s (30/30)	9.489 s (30/30)	33.719 s (30/30)
150-Queen	3.404 s (30/30)	13.632 s (30/30)	56.215 s (29/30)
160-Queen	3.594 s (30/30)	17.487 s (30/30)	60.436 s (30/30)
170-Queen	4.024 s (30/30)	18.967 s (30/30)	45.047 s (29/30)
180-Queen	4.664 s (30/30)	21.983 s (30/30)	57.396 s (22/30)
190-Queen	5.107 s (30/30)	34.322 s (30/30)	58.694 s (24/30)
200-Queen	6.180 s (30/30)	29.551 s (30/30)	78.128 s (21/30)

次に LPPH-(2) と LPPH-CSP(3) についての比較実験を表 5.2，5.3 に示す．表 5.2，5.3 により，LPPH-CSP(2) が最もよい結果を示している．これらの手法で最も時間がかかっているのは $l = 2$ の時の LPPH-CSP(3) であり，これら結果は $h_r(\boldsymbol{x})$，$s_{rij}(\boldsymbol{x})$ の計算には制約中のある 1 つの変数を用いて計算するのがよいこと示唆している．よってこれ以降では，LPPH-CSP(2) を LPPH-CSP と呼び，LPPH-CSP における 2.2 節に示した制約に対する $h_r(\boldsymbol{x})$，$s_{rij}(\boldsymbol{x})$ としては 5.1.2 節で定義した式（5.12）~（5.19）の定義式を用いる．

表 5.3　car sequencing problem に対する LPPH-CSP(2)~(3) の平均 CPU 時間

problem	LPPH-CSP(2)	LPPH-CSP(3) with $l = 1$	LPPH-CSP(3) with $l = 2$
60-01	1.393 s (30/30)	2.620 s (30/30)	8.124 s (30/30)
60-02	1.475 s (30/30)	2.202 s (30/30)	5.547 s (30/30)
60-03	1.371 s (30/30)	2.319 s (30/30)	7.784 s (30/30)
60-04	1.985 s (30/30)	3.449 s (30/30)	9.188 s (30/30)
60-05	0.913 s (30/30)	1.850 s (30/30)	7.793 s (30/30)
60-06	2.227 s (30/30)	4.516 s (30/30)	15.130 s (30/30)
60-07	0.811 s (30/30)	1.507 s (30/30)	4.431 s (30/30)
60-08	1.010 s (30/30)	1.514 s (30/30)	6.332 s (30/30)
60-09	1.545 s (30/30)	2.490 s (30/30)	10.717 s (30/30)
60-10	0.752 s (30/30)	1.217 s (30/30)	3.960 s (30/30)
65-01	1.610 s (30/30)	3.078 s (30/30)	9.920 s (30/30)
65-02	2.803 s (30/30)	4.480 s (30/30)	13.648 s (30/30)
65-03	1.548 s (30/30)	2.767 s (30/30)	10.258 s (30/30)
65-04	2.382 s (30/30)	3.431 s (30/30)	12.048 s (30/30)
65-05	1.396 s (30/30)	2.254 s (30/30)	7.336 s (30/30)
65-06	2.059 s (30/30)	2.907 s (30/30)	14.848 s (30/30)
65-07	1.177 s (30/30)	1.942 s (30/30)	6.760 s (30/30)
65-08	1.046 s (30/30)	2.011 s (30/30)	6.396 s (30/30)
65-09	1.531 s (30/30)	3.492 s (30/30)	11.405 s (30/30)
65-10	0.880 s (30/30)	1.513 s (30/30)	3.959 s (30/30)

5.2.2　LPPH-CSP の数値解析の方法

LPPH-CSP 力学系の数値計算にはオイラー法を用いた．実装に用いたオイラー法のアルゴリズムを下記に示す．式（5.1）において，x_{ij} の値が 0 または 1 の極値に近づいた場合，項 $x_{ij}(1 - x_{ij})$ は勾配ベクトルを非常に小さくさせてしまう．そのため，$x_{ij}(1 - x_{ij})$ の代わりにステップ（3）-（f）に示した手順によって x_{ij} の値を 0 から 1 の間に閉じ込める．

（1）各 i と j に対して x_{ij} の初期値としてランダムに 0 から 1 の値を生成する．

（2）各 r に対して $w_r = 0$.

（3）LPPH-CSP が CSP の解を見つけるまで次のステップを繰り返す.

(a) 各 i と j に対して $f_{ij}= w_r s_{rij}(\boldsymbol{x})$ を計算する.

(b) 各 r に対して h_r を計算する.

(c) $\Delta t = max\{f_{ij}\}/\gamma$.

(d) 各 i と j に対して $x_{ij} = x_{ij} + f_{ij}\Delta t$ を計算する.

(e) 各 r に対して $w_r = w_r + (h_r - \alpha w_r)\Delta t$ を計算する.

(f) 各 i と j に対して，もし $x_{ij} < 0$ ならば, $x_{ij} = 0$ とし, もし $x_{ij} > 1$ ならば,$x_{ij} = 1$ とする.

上記のアルゴリズムにおいて，γ はオイラー法の 1 回のステップにおける変数変化量の最大を決めるパラメータである．もし γ が小さい値をとるなら変化は小さくなり，γ が大きな値をとるなら変化は大きくなる.

図 5.2 は γ の値によって問題の解を見つけるまでの CPU 時間がどう変わるかを示した実験結果である．また，図 5.3 は γ の値によって解を見つけるまでの LPPH-CSP の探索軌跡の長さがどう変化するかを示した結果である．ここで軌跡の長さとは LPPH-CSP が与えられた初期点から解を得るまでに要した全ての変数についてのオイラー法の更新量の総和を意味する．これらの結果は LPPH-CSP を 100 個の異なる初期点から出発させ，得られた結果の平均をプロットしたものである．各試行において探索の打ち切り時間は 100[sec] とした．図 5.2 より $\gamma = 0.5$ 付近が効率的なパラメータの値であることが確認できる．また，γ の値がある程度大きくなると，LPPH-CSP は解を見つけることができなくなっている．図 5.3 から $\gamma \leq 0.5$ の場合には，LPPH-CSP の軌跡の長さはほぼ一定に保たれている．このことから，$\gamma \leq 0.5$ ならばオイラー法による LPPH-CSP の数値計算は本来の力学系による解探索能力の性質は失われていないと考えられる.

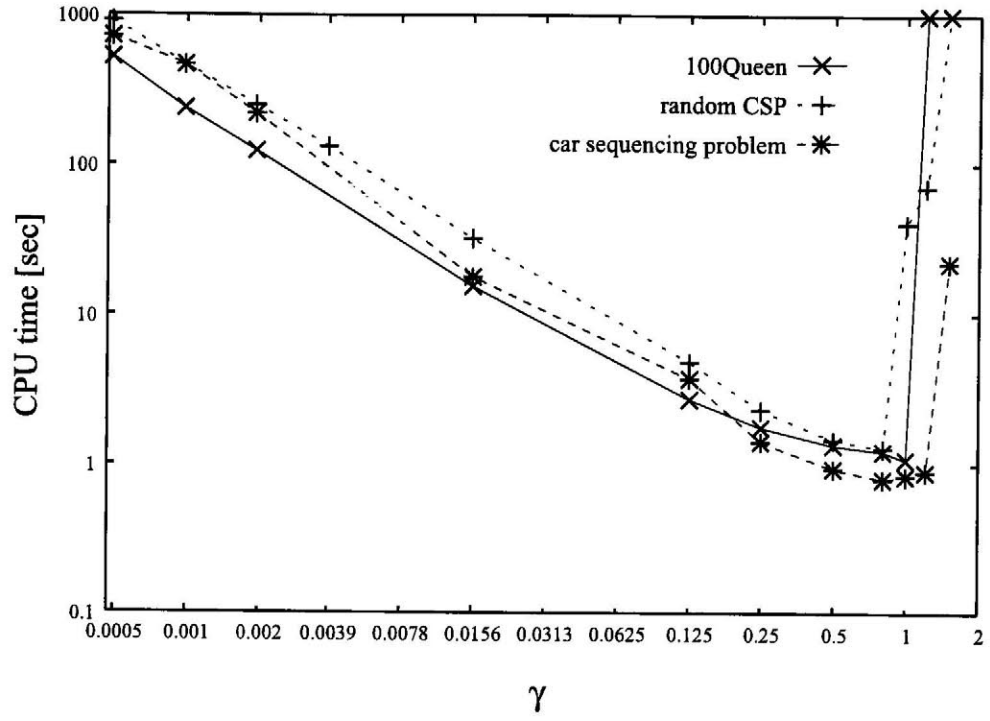

図 5.2　γに対する平均 CPU 時間の変化

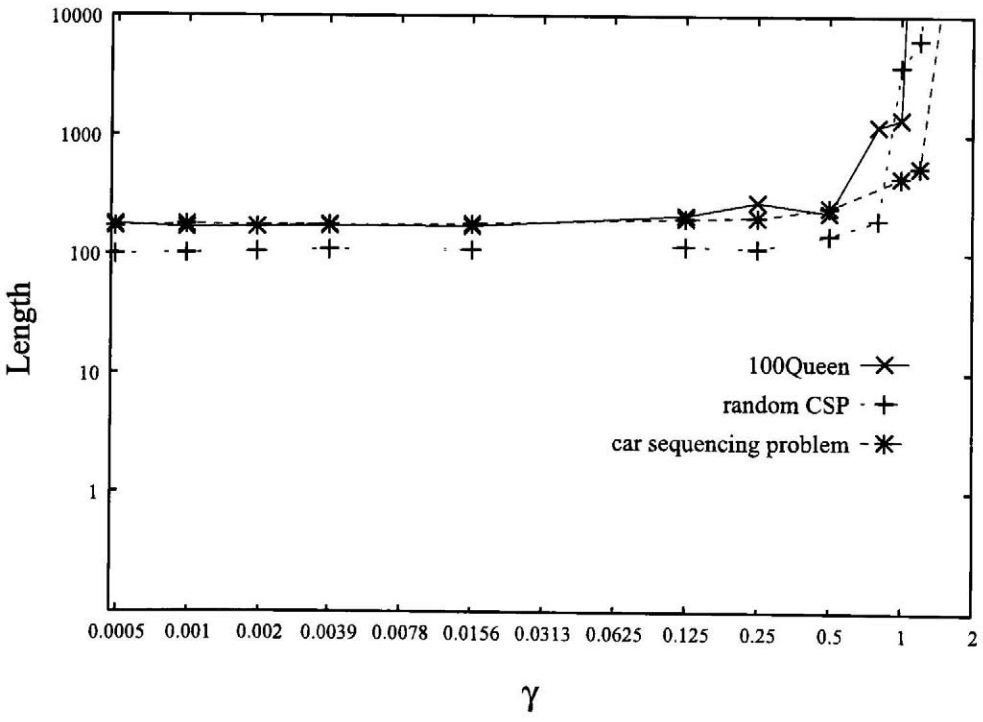

図 5.3　γに対する軌跡の長さの変化

LPPH-CSP は $\gamma \geq 0.5$ の場合は初期点によっては振動現象を起こしてしまい，解を見つけることができなくなってしまう．これは GENET の全ての変数の値を同時に更新してしまうと，振動現象を起こしてしまうため，GENET は解を見つけることができないという状況と同様の状況に陥っていると考えられる [13]．GENET は CSP の不完全解法であり，GENET の変数は離散値をとる．3.4 節で示したように GENET の変数の更新では VVP の値を 0 から 1 または，0 から 1 に反転させることにより制約違反を最小にしていく．つまり変数の値の変化量が 1 である．図 5.4 は LPPH-CSP が振動現象に陥った場合の x の探索軌跡を 2 次元平面状に射影した結果である．この結果は car sequencing problem を $\gamma = 1.5$ に設定した LPPH-CSP で解かせた結果である．探索軌跡において黒く塗りつぶされている部分は LPPH-CSP が振動現象に陥っていることを示している．図 5.5 は $\gamma = 0.5$ に設定した LPPH-CSP で同じ car sequencing problem を解かせた結果である．この場合は LPPH-CSP は振動現象に陥ることなく問題の解を見つけていることが確認できる．

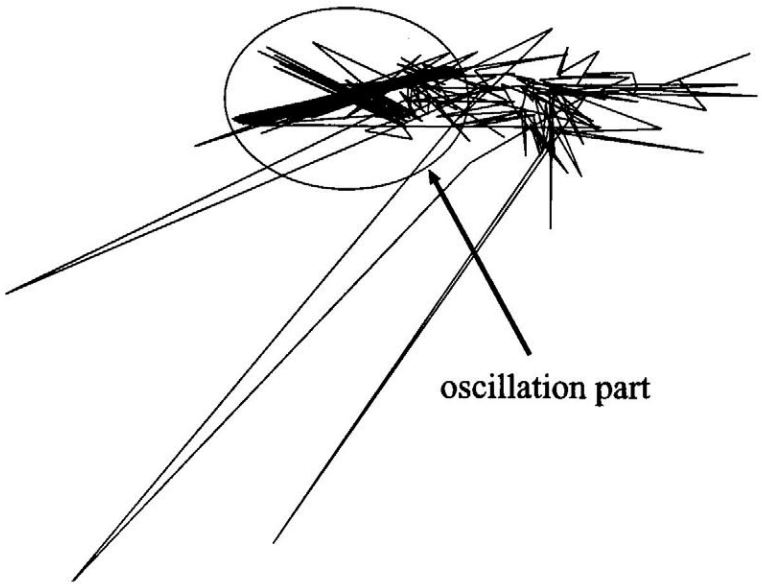

図 5.4　振動現象に陥る LPPH-CSP の軌跡 ($\gamma = 1.5$)

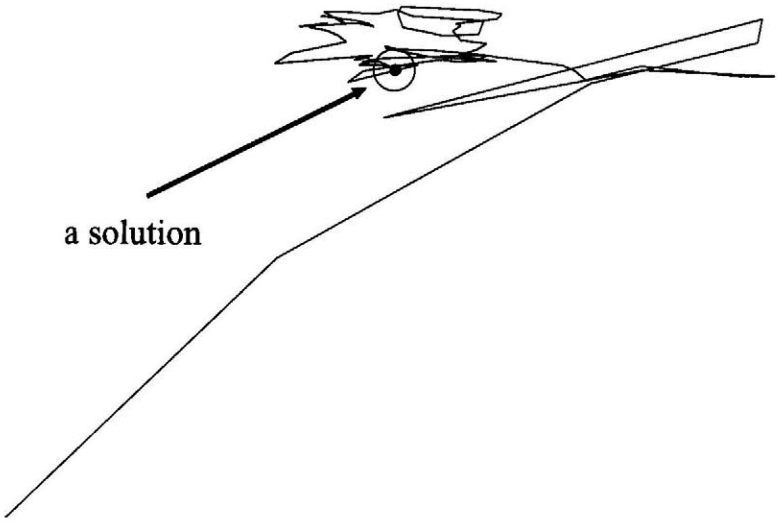

図 5.5　解に到達する LPPH-CSP の軌跡 ($\gamma = 0.5$)

また，図 5.6 に同じ問題に対して GENET を試行し，解を得るまでの軌跡を 2 次元平面上に射影したものを表す．この GENET は従来のアルゴリズムの通り変数の値を 1 個ずつ非同期に更新している．次に同じ初期値から出発し，変数の値を同時に更新した際の軌跡を図 5.7 に示す．この場合は γ が大きい場合の LPPH-CSP と同様に振動現象に陥っており，解を見つけることができなくなっていることが確認できる．このように γ が大きい場合の LPPH-CSP は GENET において変数の値を同時に更新した場合と似た挙動を示し，振動現象に陥るため，解を見つけることができない．しかし γ をある程度小さい値 ($\gamma \leq 0.5$) に設定したならば，変数の値を同時に更新しながら解を見つけ出すことができる．これは従来の離散的な CSP の解法では実現できなかった利点である．これらの結果から LPPH-CSP を数値計算する際のオイラー法で使用する γ はこれ以降 0.5 に設定する．

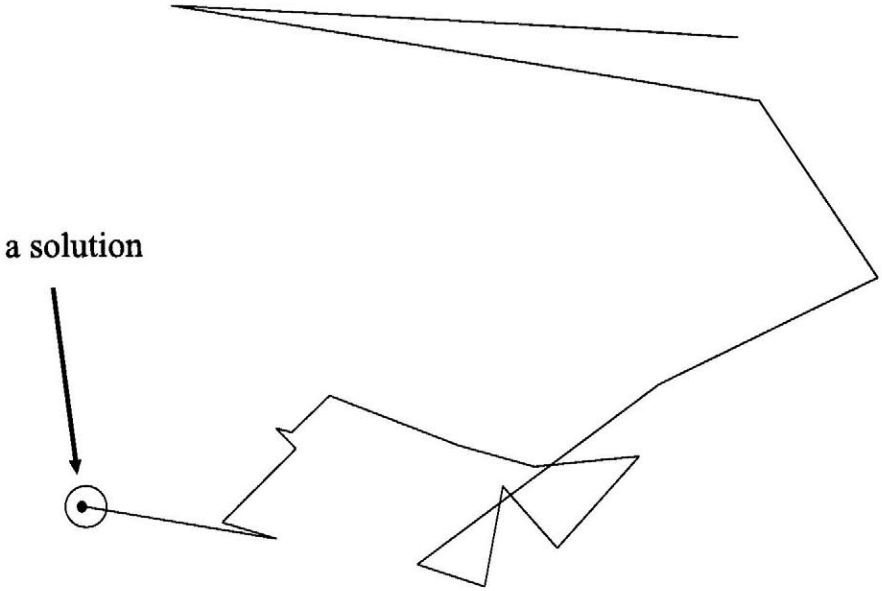

図 5.6　解に到達する GENET の軌跡 (変数の値を逐次更新)

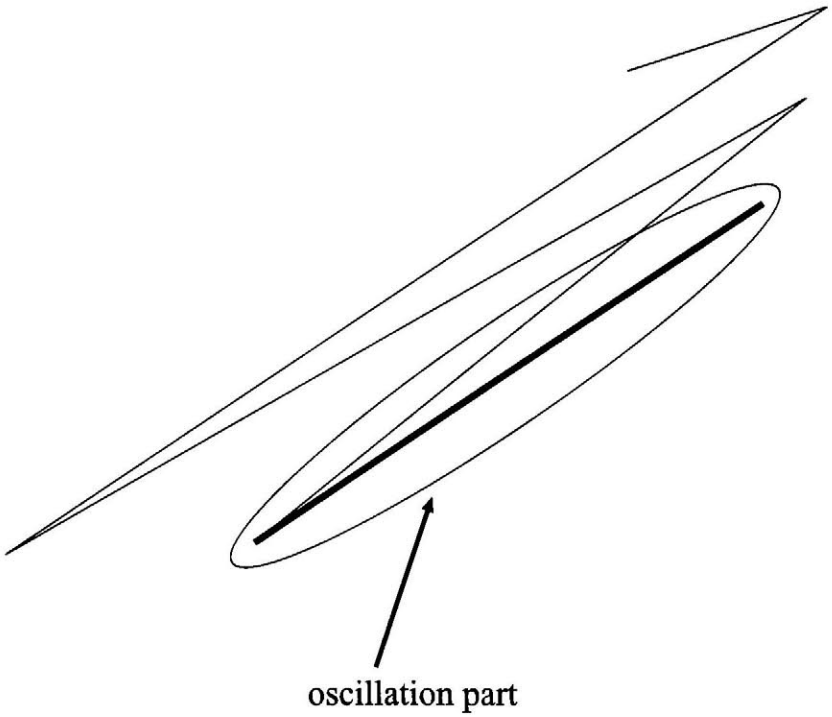

図 5.7　振動現象に陥る GENET の軌跡 (変数の値を同時更新)

5.2.3 減衰項が探索効率に及ぼす影響

LPPHについての先行研究において，解を探索するのに要するCPU時間はLPPH力学系のパラメータである減衰係数に依存することがわかっている[3]．また，SATの離散値をとる不完全解法においても減衰項に相当するパラメータを導入することにより，解探索の効率を図る研究がなされている[31]~[33]．そこで，LPPH力学系を拡張したLPPH-CSP力学系においても減衰係数が解探索の効率に影響を及ぼすかどうかを実験により調べた．結果を図5.8~5.17に示す．これらの図では縦軸は平均CPU時間（sec），横軸は減衰項αの値を表している．図5.8~5.17の結果より，LPPH-CSPにおいてもLPPHの実験結果と同様に減衰項の値が探索の効率に影響を及ぼしていることがわかる．実験結果より減衰係数の値は小さすぎてもいけないし（$\alpha = 0$付近），大きすぎても（$\alpha = 0.4$付近）時間がかかってしまう．特にこの傾向はcar sequencing problemとrandom CSPにおいて顕著であった．これらの実験結果から，設定する減衰項の値としては$\alpha = 0.1$付近がよいと考えられる．

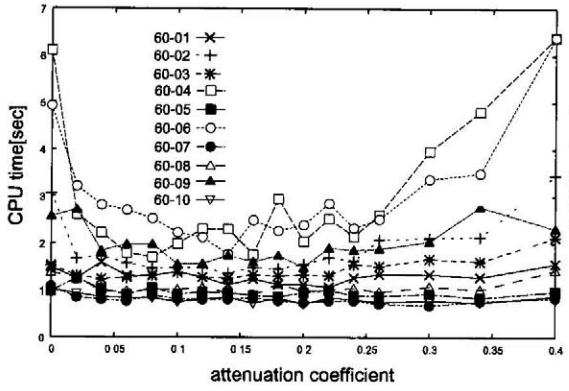

図5.8 car sequencing problem(平均利用率60%)についての結果

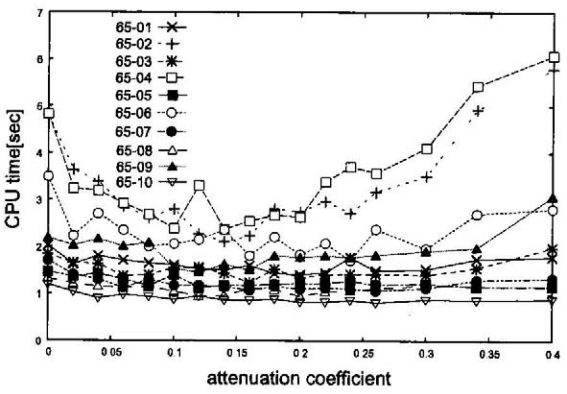

図5.9 car sequencing problem(平均利用率65%)についての結果

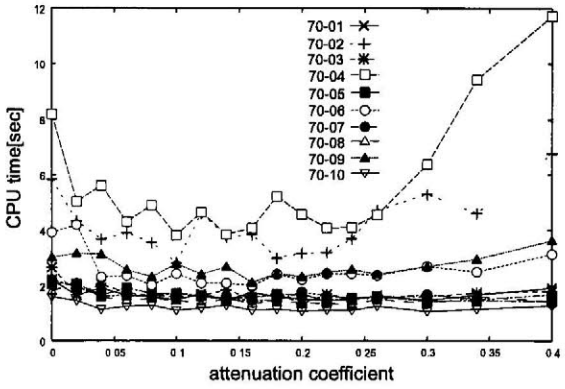

図 5.10　car sequencing problem(平均利用率 70%) についての結果

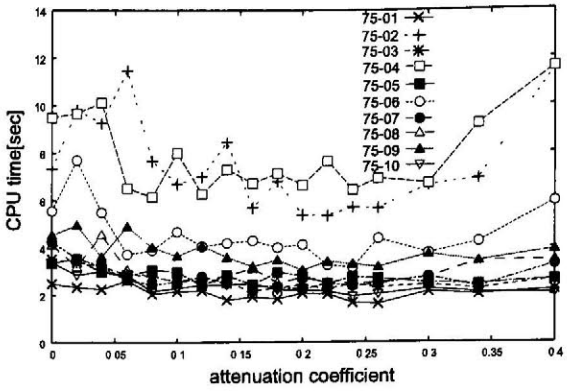

図 5.11　car sequencing problem(平均利用率 75%) についての結果

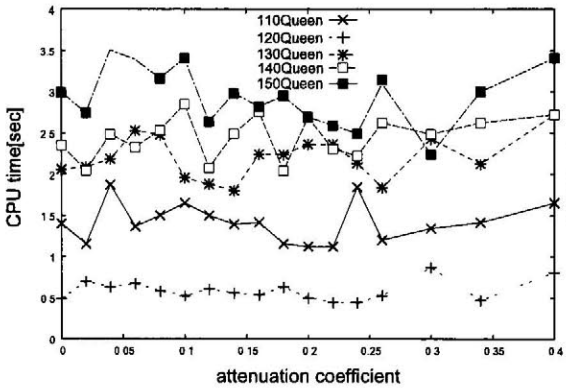

図 5.12　110~150-Queen 問題の結果

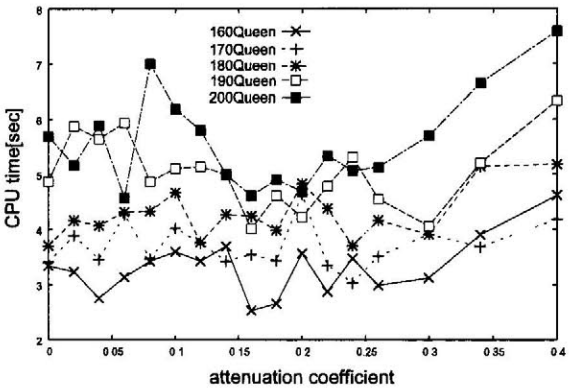

図 5.13　160~200-Queen 問題の結果

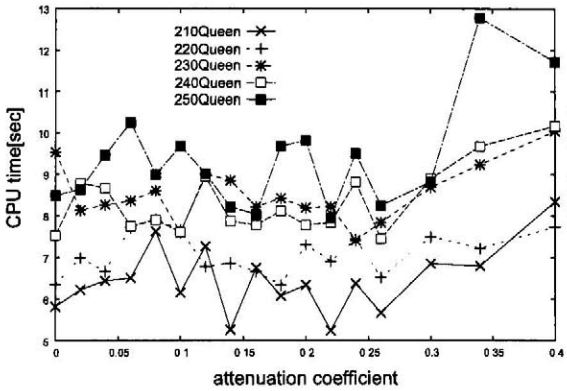

図 5.14　210~250-Queen 問題の結果

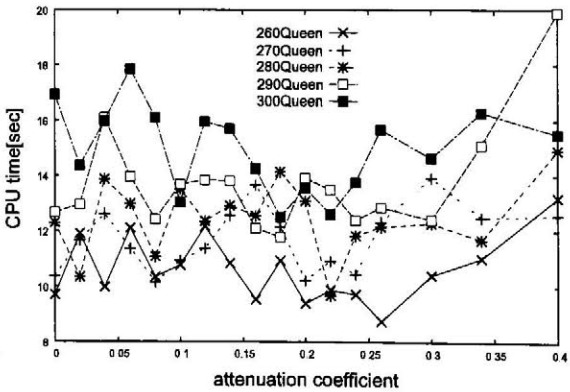

図 5.15　260~300-Queen 問題の結果

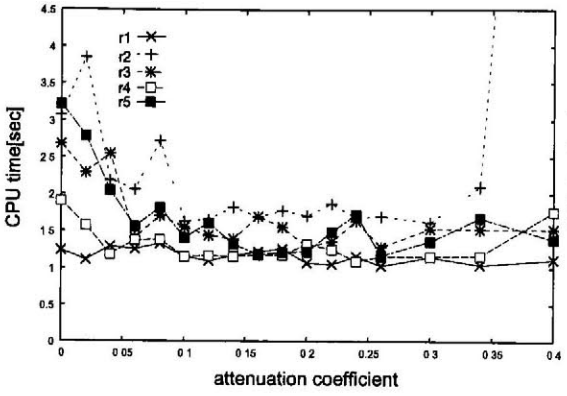

図 5.16　random CSP(変数 100 制約数 1200)についての結果

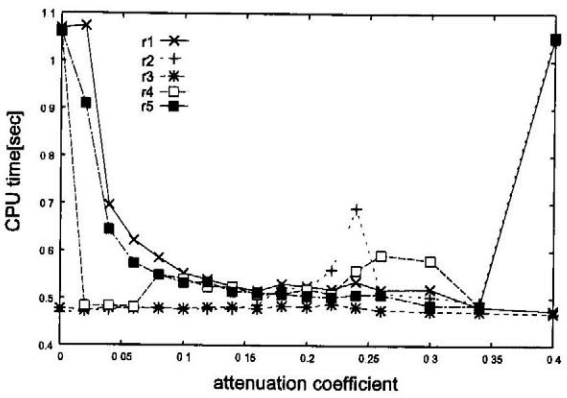

図 5.17　random CSP(変数 160 制約数 2000)についての結果

このように減衰係数の値によって解探索の効率性が変化するのはなぜだろうか? LPPH-CSP において減衰係数なしの力学系と減衰係数を導入した力学系について考えてみよう．減衰項がない場合は重みは単調増加するため，探索が進むにつれて充足のしにくい制約と充足のしやすい制約の重みの差が非常に大きくなってしまう．これによって生じる弊害のイメージを図 5.18 に示す．ここで，例として 2 つの制約 C_1 =ALT$(1, x_{11}, x_{21}, x_{32})$ と C_2 =AMT$(0, x_{11}, x_{41})$ を考える．そして探索のある時点において次のような変数割当と重みを仮定する．

$$\{x_{11}, x_{21}, x_{32}, x_{41}\} = \{1, 0, 0, 0\},$$

$$\{w_1, w_2\} = \{20, 2\}.$$

この場合，C_2 を充足させるためには x_{11} の値を小さくしなければならないのだが，C_1 の重みが非常に大きくなっているため，w_2 が w_1 より大きくなるまで制約 C_2 は非充足のままであり，効率的な探索を進める上では冗長であると考えられる．そこで，w_r の更新に減衰係数を導入することでこのような C_1 が充足しているならば w_1 は小さくなり，そのことで効率的に制約解消の競合が図れているため，実験では減衰係数 $\alpha = 0$ のときよりもある程度の大きさの α を力学系に導入した方が早く問題の解を見つけていると考えられる．減衰係数ありの場合とない場合の上記の例の重み変化のイメージを図 5.19 に示す．

次に減衰係数が大きい場合について考える．前述で示した制約 C_1, C_2，変数割当を仮定する．この場合，減衰係数が大きいと，C_1 が充足すると重み w_1 はすぐに 0 に近づく，そのため C_2 は x_1 によって充足するが，すぐに w_2 も小さくなるため，x_1 はまた大きくなり，C_2 は非充足の状態に陥る．このように減衰係数が大きいと制約間での充足/非充足の振動が頻繁

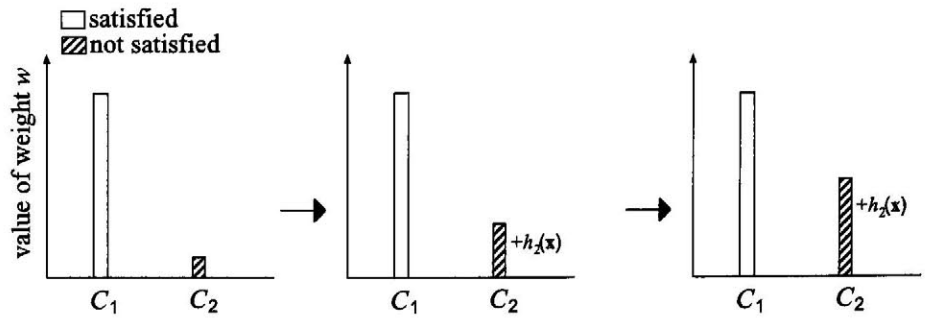

図 5.18　重み変化のイメージ図（減衰係数なし）

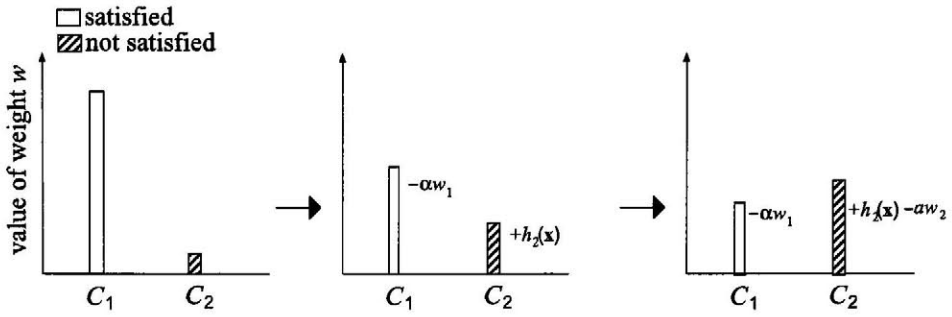

図 5.19　重み変化のイメージ図（減衰係数あり）

に起こってしまうため，解の探索効率が悪くなってしまう．しかし，減衰係数がないあるいはある程度小さい値なら C_2 が充足しても重みはすぐには小さくならないので x_1 は 0 にする力が働き，x_{21} または x_{32} によって制約充足をおこなうことができる．

5.2.4　GENET との比較

LPPH-CSP について解探索の有効性を調べるために CSP の効率的な解法として知られている GENET との比較実験を行った．図 5.20~5.28 がその結果である．各図は各問題に対する GENET，LPPH-CSP の平均 CPU 時間（sec）を示している．実験では減衰項として $\alpha = 0.1$ を LPPH-CSP に用いた．比較実験の結果より LPPH-CSP は GENET と同等もしくはそれ以上の性能を確認することができる．これらの実験結果はシングル CPU の計算機実験によって得られた結果である．実際は LPPH-CSP の変数の並列実行は実現していない．LPPH-CSP はニューラルネットワークであり，図 5.1 に示されるように，変数ニューロン x_{ij} と制約ニューロン C_r は計算素子であり，それらは並列に実行することができる．GENET も離散的なニュー

ラルネットワークとして提案されており[12],[13]，3.4.4節の図3.6に示されるようにGENETでも同様にVVPと制約は計算素子である．しかし，GENETでは全ての計算素子を同時に実行することはできない[13]．一方，LPPH-CSPでは実験結果が示したように同時に実行することが可能である．そのため，FPGAなどのハードウェアでLPPH-CSPを実装すれば，GENETなどの変数の並列更新を行うことができない解法をハードウェアに実装したものに比べ，解探索能力の著しい向上を期待することができる．このようにGENETとの比較実験結果はLPPH-CSPの有効性を明らかにした．

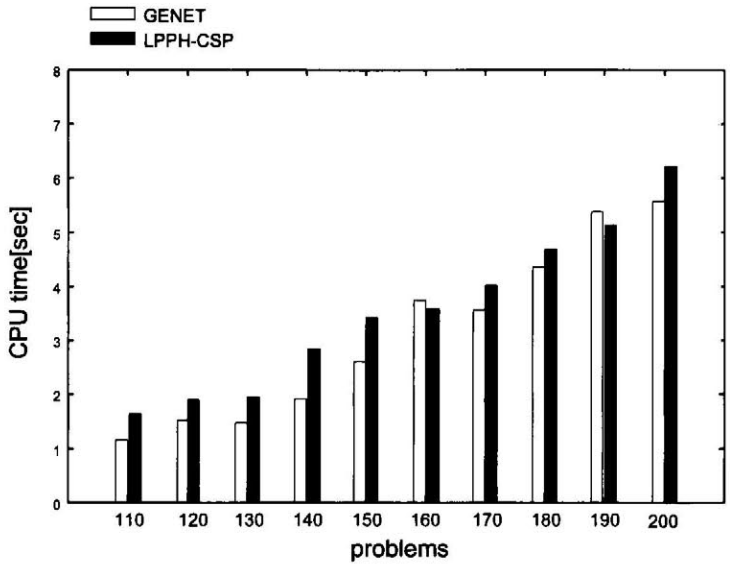

図 5.20　110~200-Queen 問題の比較結果

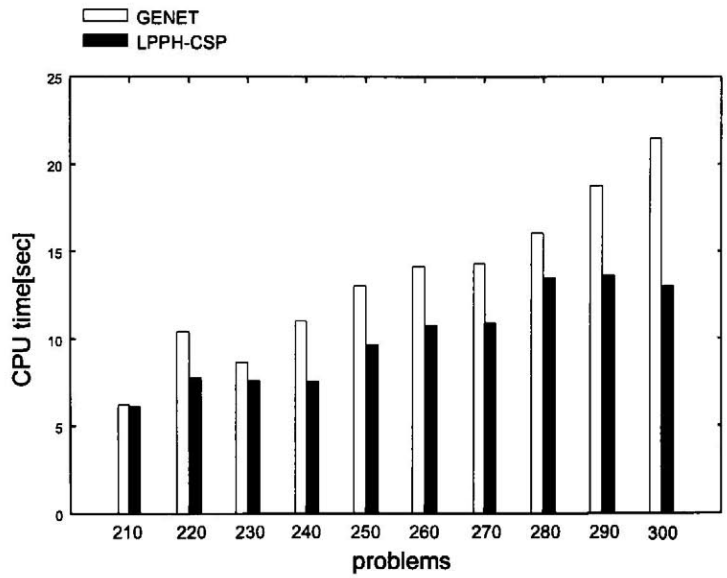

図 5.21　210~300-Queen 問題の比較結果

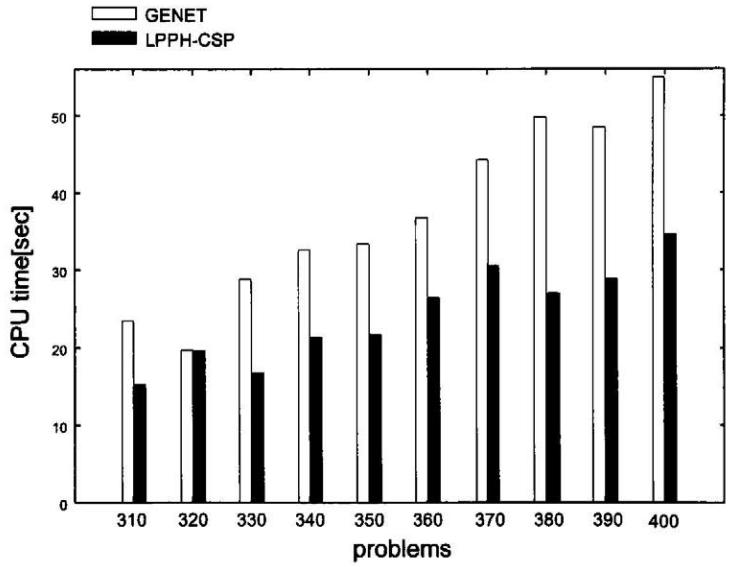

図 5.22　310~400-Queen 問題の比較結果

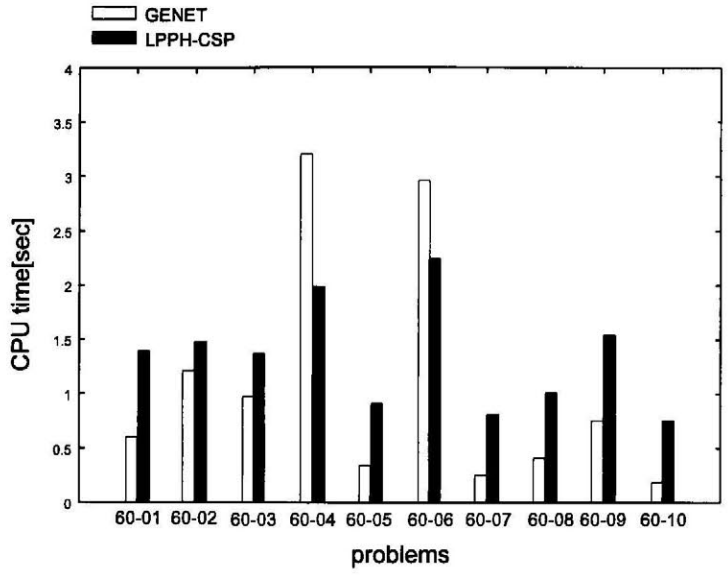

図 5.23　car sequencing problem（平均利用率 60%）の比較結果

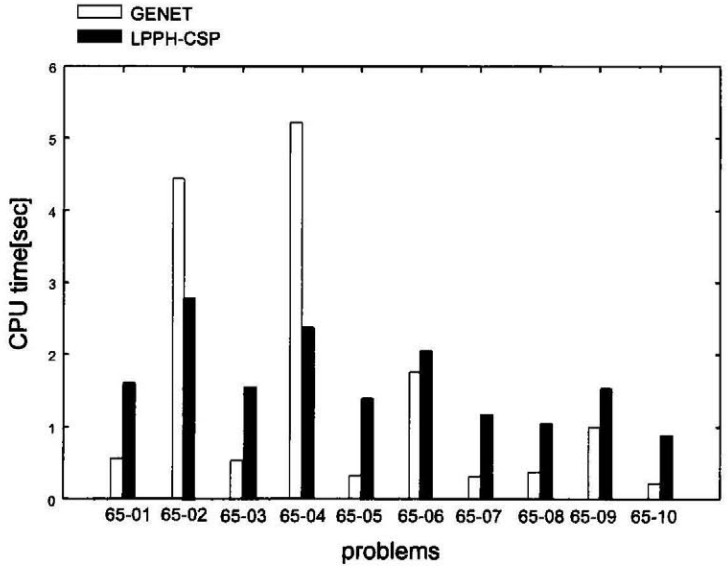

図 5.24　car sequencing problem（平均利用率 65%）の比較結果

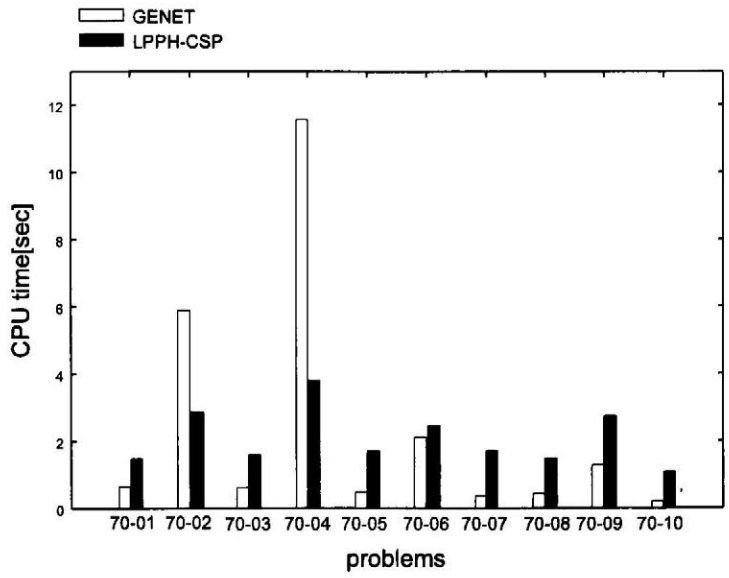

図 5.25　car sequencing problem（平均利用率 70%）の比較結果

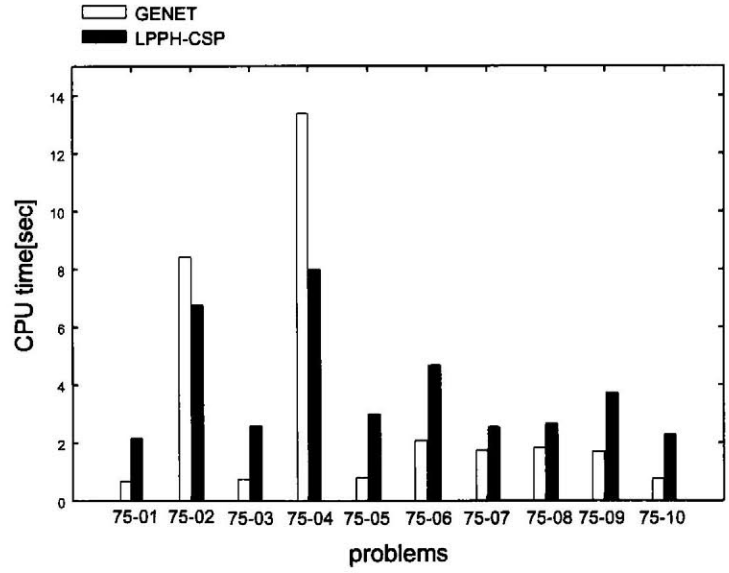

図 5.26　car sequencing problem（平均利用率 75%）の比較結果

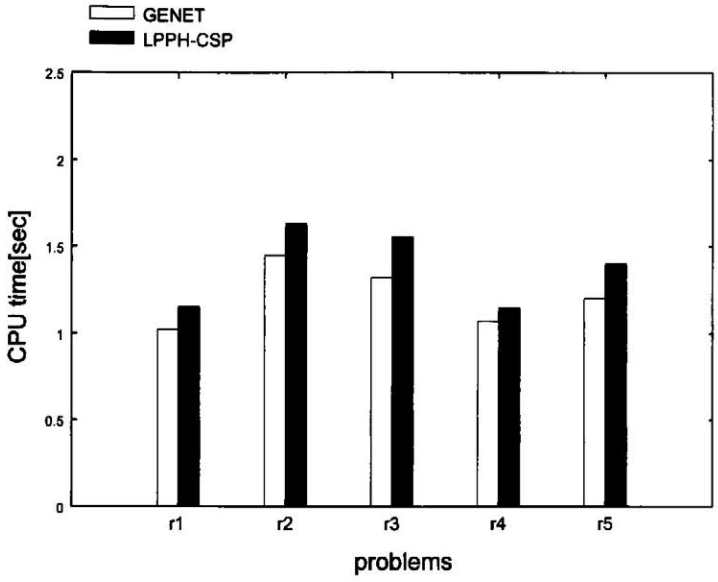

図 5.27　random CSP(変数 100 制約数 1200) の比較結果

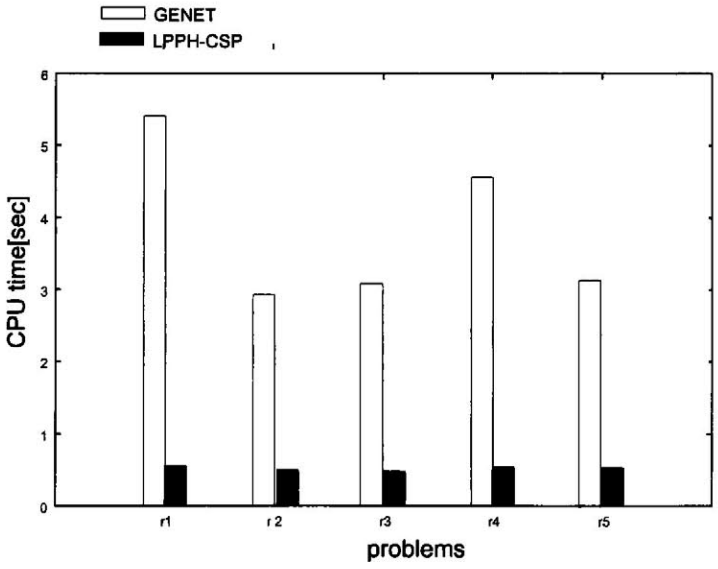

図 5.28　random CSP(変数 160 制約数 2000) の比較結果

5.3　冗長な制約が探索の効率に及ぼす影響について

問題を 2.2 節で述べた一般的な制約で表現する際に，必要最小限の制約で表現するよりも，冗長な制約を加えて表現した問題に対して LPPH-CSP を適用した方が早く解を探索できる場合がある．ここで冗長な制約とはその制約を加えても元の問題の解の個数を変えることは

表 5.4　car sequencing problem の制約表現（必要最小限の制約表現）

制約の種類	制約が表現しているもの
ALT ∧ AMT	各車はあるタイプに属する
AMT	オプション i をもつ車を N 台生産する
AMT	各作業領域での容量制約

表 5.5　car sequencing problem の制約表現（冗長な制約を含む制約表現）

制約の種類	制約が表現しているもの
ALT ∧ AMT	各車はあるタイプに属する
ALT ∧ AMT	オプション i をもつ車を N 台生産する
AMT	各作業領域での容量制約

ない制約のことである．例えば，car sequencing problem を LPPH-CSP を用いて解を求める際には，その問題を必要最小限の制約で表現するよりも，冗長な制約を加えた形で問題を表現した方が，解探索の効率性が向上することが実験などから確認された．表 5.4，5.5 にそれぞれ car sequencing problem を表現する際に必要最小限の制約表現，冗長な制約を加えた制約表現を表している．表 5.5 において 2 行目の ALT 制約が冗長な制約である．ALT 制約，AMT 制約を用いて car sequencing problem における生産制約を表現しているが，実際は表 5.4 のように生産制約は AMT 制約のみで表現することができる．

これら 2 つの制約表現によって記述した各 car sequencing problem に LPPH-CSP を適用して，解を見つけるまでの平均 CPU 時間（sec）を表したものを図 5.29~5.32 に示す．

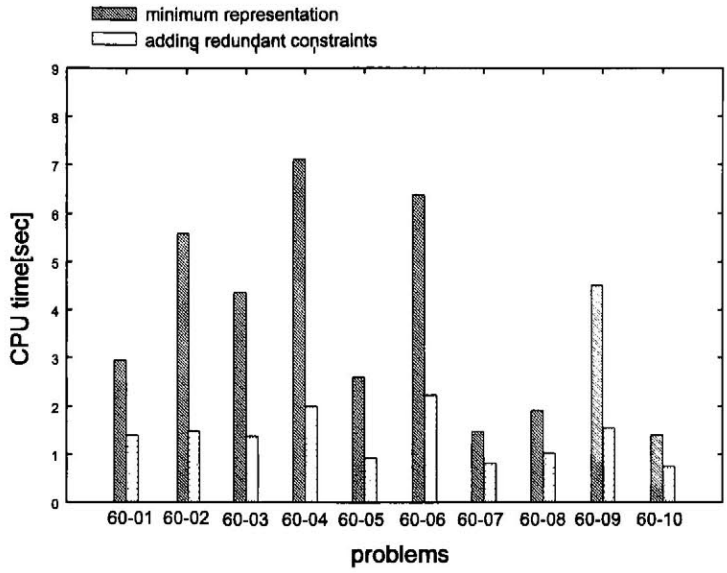

図 5.29　car sequencing problem（平均利用率 60%）に対する冗長な制約を加えたことによる探索時間の変化

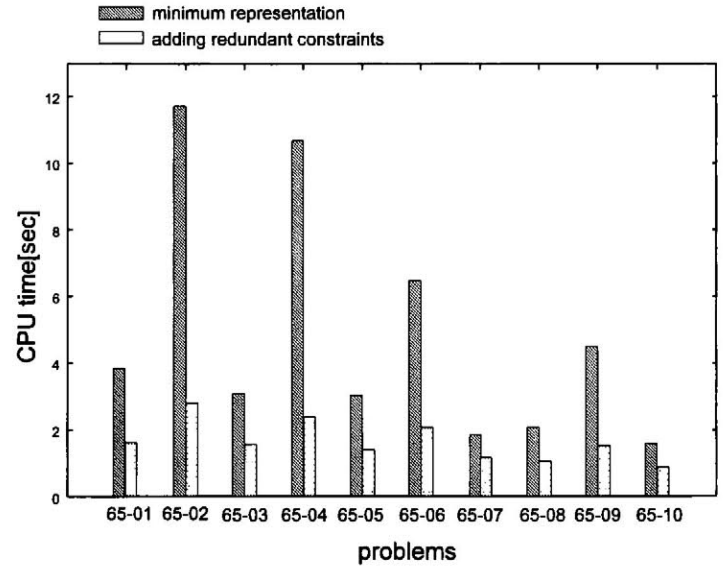

図 5.30　car sequencing problem（平均利用率 65%）に対する冗長な制約を加えたことによる探索時間の変化

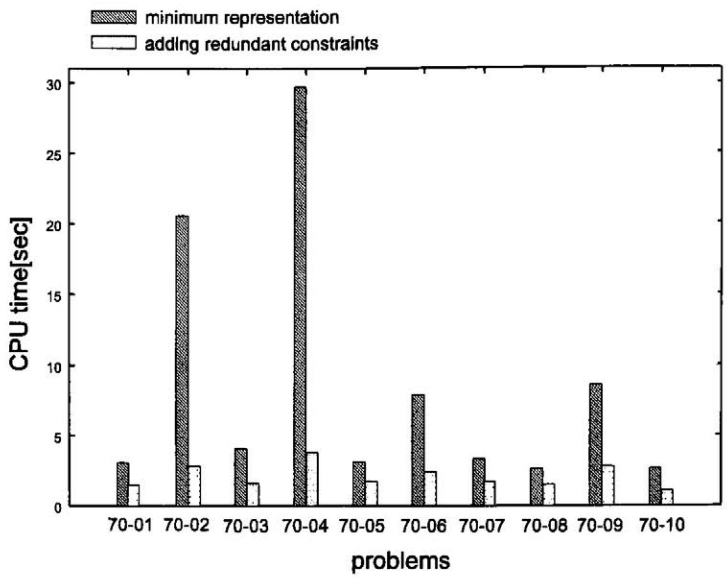

図 5.31　car sequencing problem（平均利用率 70%）に対する冗長な制約を加えたことによる探索時間の変化

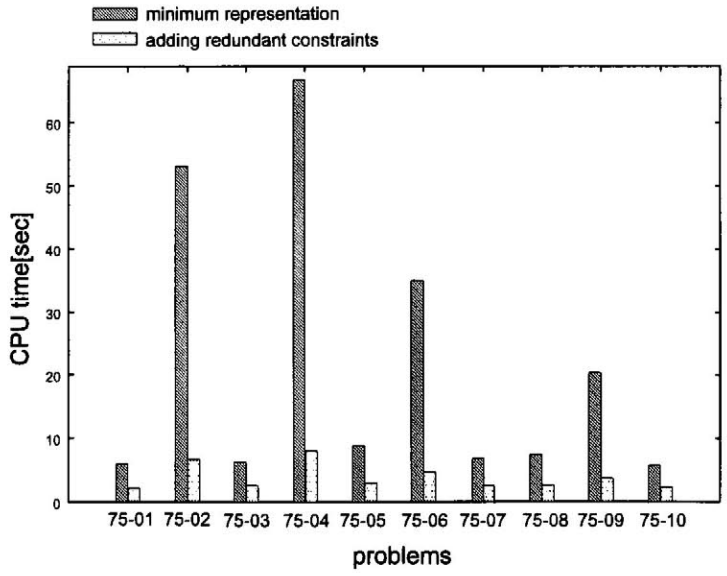

図 5.32　car sequencing problem（平均利用率 75%）に対する冗長な制約を加えたことによる探索時間の変化

これらの結果から冗長な制約を加えた問題を解いた場合の方が効率的に解を探索していることが確認できる．しかし，どの問題に対しても冗長な制約を加えたほうがよい結果を得られるわけではない．表 5.6，5.7 に N-Queen Problem を表現する際に必要最小限の制約表現と冗長な制約を加えた制約表現を示す．表 5.7 において 2 行目の ALT 制約が冗長な制約に対応する．そして，図 5.33，5.34 にこれらの制約表現による各 N-Queen 問題を LPPH-CSP

で解いた際の結果を示す.

表 5.6　N-Queen 問題の制約表現（必要最小限の制約表現）

制約の種類	制約が表現しているもの
ALT ∧ AMT	盤目の各行には必ず 1 つ Queen が置かれる.
AMT	盤目の各列には 1 つ以上 Queen は置かれない.
AMT	盤目の対角線には 1 つ以上 Queen は置かれない.

表 5.7　N-Queen 問題の制約表現（冗長な制約を含んだ制約表現）

制約の種類	制約が表現しているもの
ALT ∧ AMT	盤目の各行には必ず 1 つ Queen が置かれる.
ALT ∧ AMT	盤目の各列には 1 つ以上 Queen は置かれない.
AMT	盤目の対角線には 1 つ以上 Queen は置かれない.

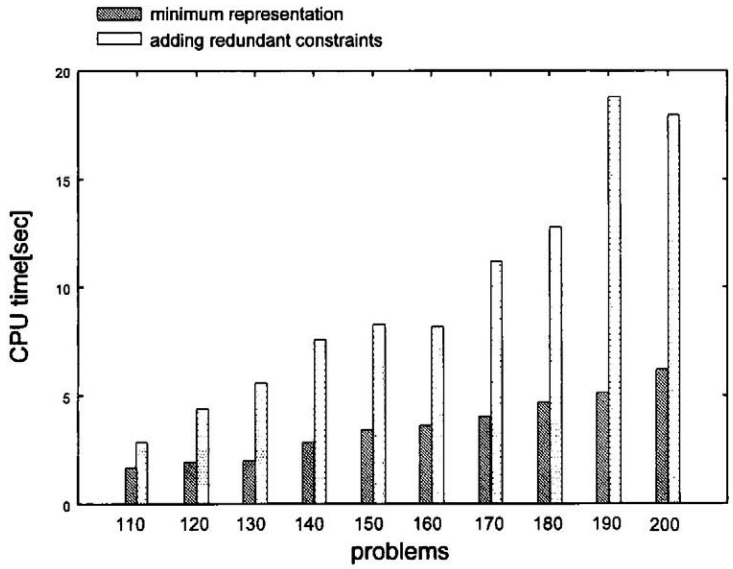

図 5.33　110~200-Queen 問題に対する冗長な制約を加えたことによる探索時間の変化

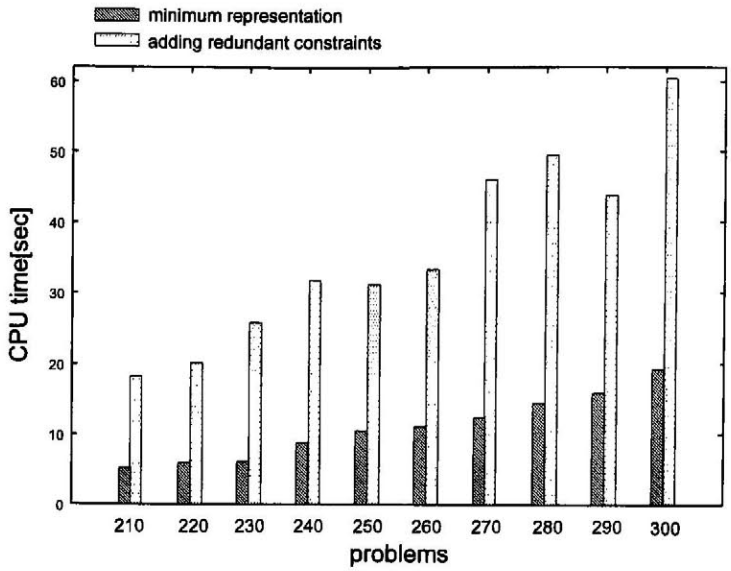

図 5.34　210~300-Queen 問題に対する冗長な制約を加えたことによる探索時間の変化

これらの結果は冗長な制約を加えてた場合よりも最小限の制約表現で記述された問題を解いた方が解を早く見つけ出していることを示している．直観的に考えると，冗長な制約を加えることは加えた数だけ余分な $h_r(\boldsymbol{x})$, $s_{rij}(\boldsymbol{x})$ を計算しなければならないので，計算が余計にかかってしまう．しかし，car sequencing problem を解く場合には，冗長な制約を加えた制約表現で記述した問題の方が LPPH-CSP は早く解を見つけ出していることが実験から確認された．この加えた冗長な制約は元の問題の容量制約を表現しているが．この制約は問題を解く際には重要であるので，効率的な解探索を行うためには冗長な制約が効果的な情報を与えているのではないかと考えられる．

5.4　制約の重要度を考慮した手法

前節で示したように，どのように問題の制約を記述するかは解を探索する時間に大きな影響を与える．しかし，ある問題に対して LPPH-CSP が効果的に解を探索することができる制約の最適な記述表現をあらかじめ調べるのは難しい．問題の制約をどう記述するかということは，冗長な制約を追加することにより問題の情報をより多く与えることのほかにどの制約をどのくらい重要視するかということも意味する．そこで LPPH-CSP において制約の重要度を考慮した項を導入することを提案する．重要度を考慮するように式（5.1）の力学系

を次式にようにに拡張する [17].

$$\frac{dx_{ij}}{dt} = x_{ij}(1-x_{ij})\sum_{r=1}^{m}\rho_r w_r s_{rij}(\boldsymbol{x}), \text{ for all VVP } x_{ij}, \tag{5.28}$$

$$\frac{dw_r}{dt} = h_r(\boldsymbol{x}) - \alpha w_r,\ r = 1,2,\cdots,m. \tag{5.29}$$

上記の力学系を LPPH-CSP with IC（LPPH-CSP with Importance of Constraints）と呼ぶ．式（5.28）では，制約 C_r の重要度を表す項 ρ_r を導入した．ρ_r は事前に獲得した情報もしくは，探索を進めていく上でなんらかのヒューリスティックによって獲得した情報によって決定する．ここではまず重要度をあらかじめ設定することにより解の効率化を図ることが出来るかどうかを調べる．表 5.8 に各制約に対して設定した ρ_r の値，図 5.35~5.38 にその ρ_r を用いた LPPH-CSP with IC と従来の LPPH-CSP を car sequencing problem に適用した結果を示す．car sequencing problem の制約の記述には必要最小限の制約記述を用いている．

表 5.8　car sequecing problem において設定した ρ_r の値

制約の種類	制約が表現しているもの	ρ_r の値
ALT	各車はあるタイプに属する	1.0
AMT	各車はあるタイプに属する	2.0
AMT	オプション i をもつ車を N 台生産する	2.0
AMT	各作業領域での容量制約	2.0

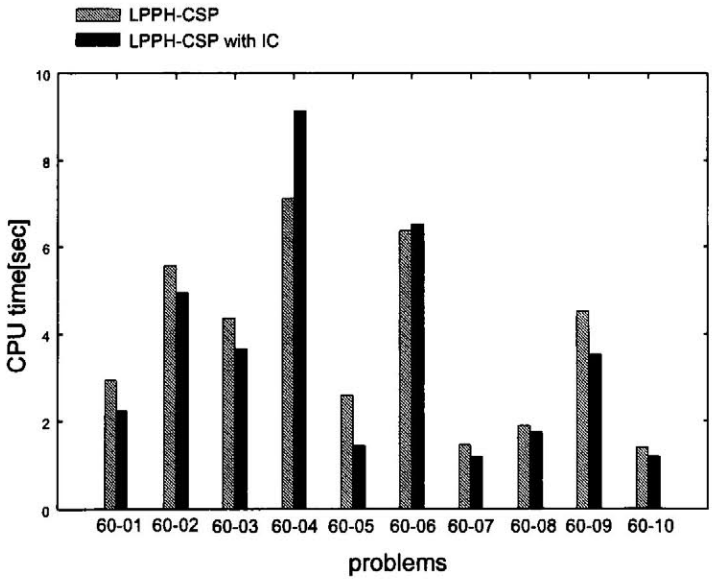

図 5.35　car sequencing problem（平均利用率 60%）に対する LPPH-CSP と LPPH-CSP with IC の比較実験結果

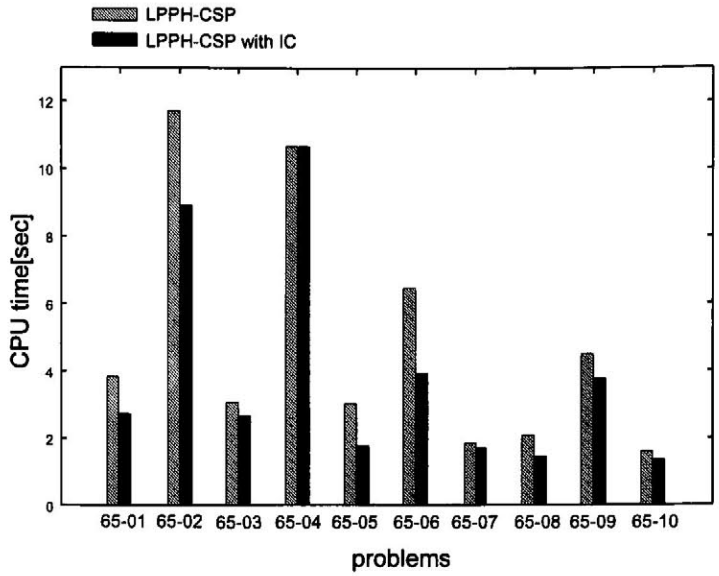

図 5.36　car sequencing problem（平均利用率 65%）に対する LPPH-CSP と LPPH-CSP with IC の比較実験結果

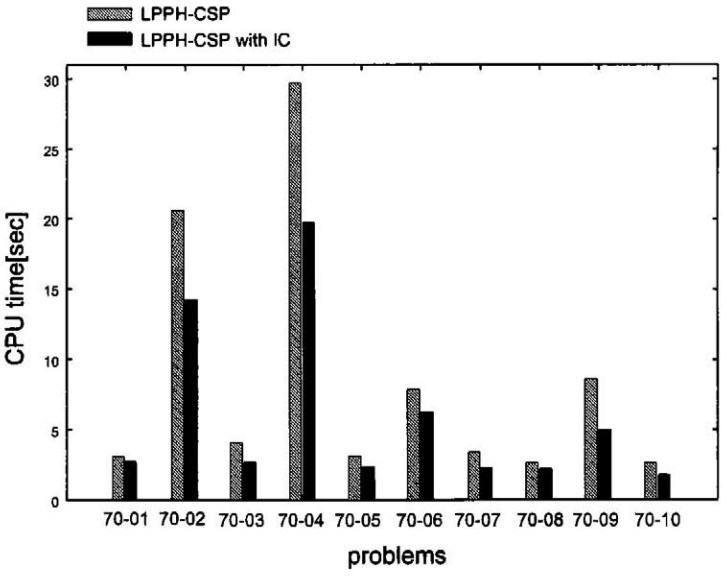

図 5.37　car sequencing problem（平均利用率 70%）に対する LPPH-CSP と LPPH-CSP with IC の比較実験結果

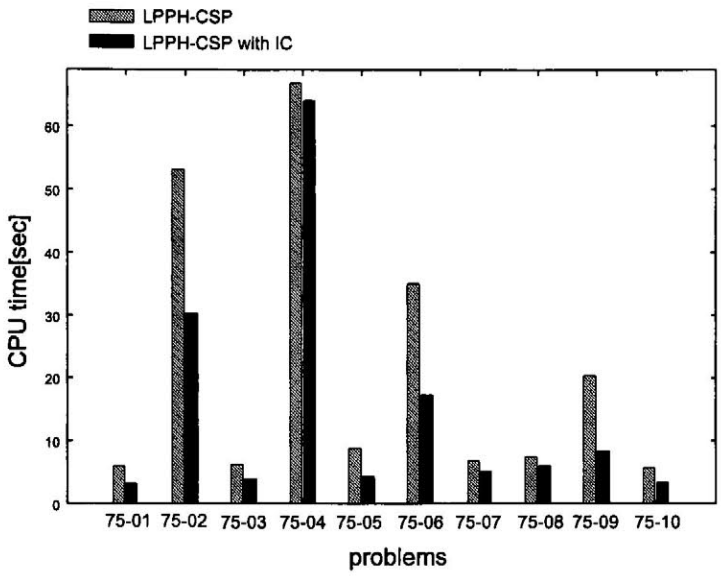

図 5.38　car sequencing problem（平均利用率 75%）に対する LPPH-CSP と LPPH-CSP with IC の比較実験結果

図 5.35~5.38 の結果からほとんどの問題に対して LPPH-CSP with IC は LPPH-CSP よりも早く問題の解を見つけていることがわかる．しかし，前節で示した冗長な制約を追加した場合よりはよい結果を得ることができなかった．表 5.8 で示した ρ_r の値は実験により経験的に求めた．今後の課題としては，ρ_r を LPPH-CSP with IC を適用する前に自動的に設定する手法や探索の途中で動的に ρ_r を求める手法への発展が挙げられる．もし，ヒューリスティックにより，探索を進めていく上で，ρ_r の値を動的またはあらかじめ自動的に重要度を決定できるのならば，問題の最適な制約表現を試行錯誤的に調べることなしに，効率的な探索を期待することが出来る．

第6章　目的関数を持つ制約充足問題を解くニューラルネットワーク LPPH-OCSP

5章ではSATの解法であるLPPHをCSPを解くために拡張を行った．CSPは与えられた制約を全て充足する解も見つけだす問題であるが，実問題での応用を考える場合ならば，満たすべき制約とともに何らかの目的（コストを最小化するなど）が与えられる場合も考えられる．本節ではこのような問題に対処するために，目的関数付きCSP(OCSP)を定義し，その問題を解くためにLPPH-CSPを拡張する[18]．

6.1　目的関数と線形不等式制約を持つ制約充足問題

目的関数を持つCSP（Constraint Satisfaction Problem with Objective function; OCSP)を次のように定義する．

(OCSP)　find $\boldsymbol{x}$,

such that minimize $E(\boldsymbol{x})$,

subject to $\{C_r \mid r = 1, 2, .., m\}$. (6.1)

$E(\boldsymbol{x})$は問題によって与えらる目的関数である．上記の定義のように，OCSPの解は全ての制約を充足し，かつ目的関数$E(\boldsymbol{x})$を最小化するような変数への値の割当である．

OCSPの制約としては2.2節で述べた一般的な制約と線形の不等式制約を持つものとする．線形の不等式C_rは次式のように定義される．

$$\sum c_{rij} x_{ij} \leq \sigma_r. \tag{6.2}$$

c_{rij}はC_rに現れるVVP x_{ij}の係数であり，σ_rは定数である．

6.2 Warehouse Location Problem

WLP (Warehouse Location Problem)[34] は OCSP で表現することができる問題の 1 つであり，資源割当問題の代表的な問題として知られている．WLP において，会社は販売店に商品を供給するために，幾つかの候補地から卸売店を開店しなければならない．卸売店を候補地から開店した場合，維持コストがかかり，売店に品物を供給する際には供給コストがかかる．また，売店からの商品の要求に対して各卸売店が供給することができる限界を示す容量を卸売店は持っている．WLP の目的は維持コストと供給コストの総和が最小となるに，全ての店に品物を供給するように卸売店を開店することである．WLP の制約は次のように記述される．

- 会社は候補地から卸売店を開店する．
- 各売店はちょうど 1 つの卸売店から品物の供給を受ける．
- 各卸売店は品物を供給することができる限界値をもっている．

WLP の目的関数 $E(\boldsymbol{x})$ は次式によって表現される．

$$E(\boldsymbol{x}) = \sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}c_{ij} + \sum_{j=1}^{m} x_{j0}d_j. \tag{6.3}$$

上式において c_{ij} は売店 i が卸売店 j から供給を受ける際に要する供給コストである．d_j は卸売店 j が開店する場合に要する維持コストである．x_{ij} は“売店 i が卸売店 j から供給を受ける”ことを表すブール変数であり，x_{j0} は“卸売店 j が開店する”ことを表すブール変数である．また n，m はそれぞれ売店と卸売店の数を表している．図 6.1 は WLP の単純な例を示している．図 6.1 において各売店は卸売店 1 または卸売店 2 から供給を受ける．品物を供給することができる限界の容量は卸売店 1 は 6，卸売店 2 は 4 である．また，各卸売店は開店または閉店の状態をとる．もし閉店するならば卸売店は売店に品物を供給することができない．この場合は，卸売店の容量は 0 となる．この WLP の例を CSP で表現したものを図 6.2 に示す．

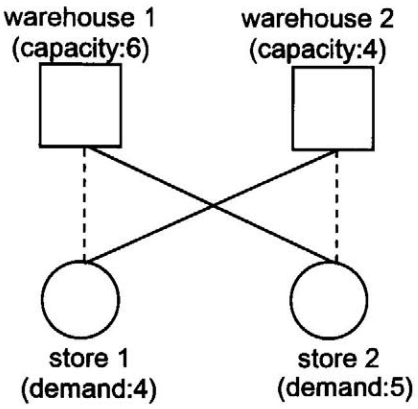

図 6.1　WLP（2 卸売店，2 売店）の例

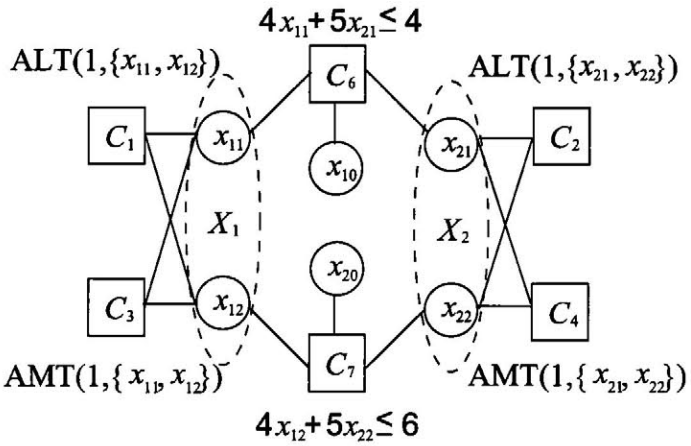

図 6.2　CSP による WLP の表現

6.3　OCSP を解くニューラルネットワーク LPPH-OCSP

6.3.1　LPPH-OCSP の力学系

我々は OCSP を解くために次式で示されるように LPPH-CSP を拡張する．本論文では OCSP で表現することができる WLP への適用を想定しており，目的関数 $E(\boldsymbol{x})$ は線形の目的関数と

する．

$$\frac{dx_{ij}}{dt} = x_{ij}(1-x_{ij})\left(\sum_{r=1}^{m} W_r w_r x_{ij} - \delta L \frac{\partial F(\boldsymbol{x})}{\partial x_{ij}}\right), \tag{6.4}$$

$$\frac{dw_r}{dt} = h_r(\boldsymbol{x}) - \alpha w_r, \tag{6.5}$$

$$\frac{dW_r}{dt} = h_r(\boldsymbol{x}) - \beta W_r, \tag{6.6}$$

$$\frac{dL}{dt} = (1-L)L\left(\mathrm{Max}\left\{0, 1-\sum_{r=1}^{m} h_r(\boldsymbol{x})\right\}\mu - \epsilon L\right). \tag{6.7}$$

この力学系を LPPH-OCSP と呼ぶ．ここで $F(\boldsymbol{x})$ は目的関数 $E(\boldsymbol{x})$ を正規化した関数である．α と β は減衰係数である（$\alpha > \beta$）．つまり，w_r は制約 C_r の短期的な充足状況を記憶する重みであり，W_r は C_r の長期的な充足状況を記憶する重みとなる．L は制約の充足状況により制約充足と目的関数最小化のどちらを重要視するかを調整する変数である．全ての制約が充足するならば，L を大きくすることにより，変数は目的関数をより最小化するように変化し，現在の解から離れるように動作する．また δ，μ，ϵ はパラメータである．

線形不等式 C_r に対する非充足度関数 $h_r(\boldsymbol{x})$ と充足指示関数 $s_{rij}(\boldsymbol{x})$ は次のように定義される．

$$h_r(\boldsymbol{x}) = \mathrm{Max}\left\{0, \sum c_{rij} x_{ij} - \sigma_r\right\}, \tag{6.8}$$

$$s_{rij}(\boldsymbol{x}) = h_r\left(\boldsymbol{x}^{[ij,0]}\right) - h_r\left(\boldsymbol{x}^{[ij,1]}\right), \tag{6.9}$$

$$\text{where } x_{kl}^{[ij,0]} = \begin{cases} 0, & \text{if } (k,l) = (i,j), \\ x_{kl}, & \text{otherwise,} \end{cases} \quad \text{and}$$

$$x_{kl}^{[ij,1]} = \begin{cases} 1, & \text{if } (k,l) = (i,j), \\ x_{kl}, & \text{otherwise.} \end{cases} \tag{6.10}$$

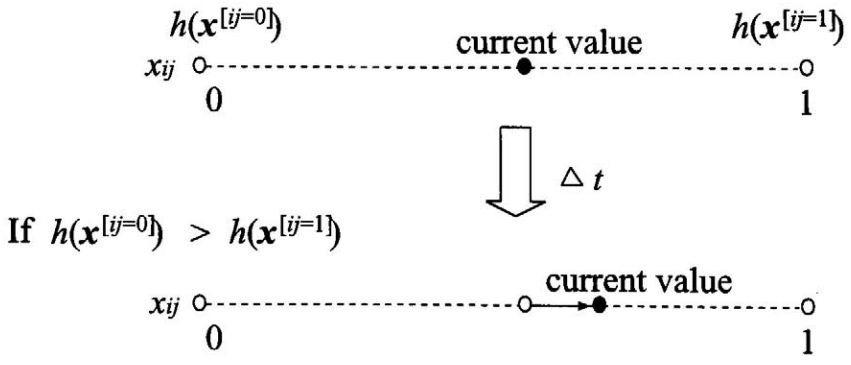

図 6.3　線形不等式を充足させるための変数の値の変化

式（6.9），（6.10）によって定義される充足指示関数では，図 6.3 で表されるように各変数はその変数が 0 になったときの非充足度関数 $h_r\left(\boldsymbol{x}^{[i,0]}\right)$ と 1 になったときの非充足度関数 $h_r\left(\boldsymbol{x}^{[i,1]}\right)$ を計算し，より制約を充足させる方に変数を変化させることにより，線形の不等式を充足させる．CSP を LPPH-OCSP を適用して解く場合，離散空間である解空間を連続空間に緩和する．このため，上記の定義では 0 から 1 の間の連続値をとる VVP によって線形不等式の制約が充足してしまう可能性がある．しかし，2.1 節の定義で述べたように CSP が“各変数はその変数が持つ変域の中から唯一つだけ値としてとる”という内在する制約をもっているため，この制約を満たす限り，全ての VVP は 0 か 1 の離散値をとる．そのため，全ての制約が充足されるのならば，線形不等式制約も連続値によって充足されることはない．つまり CSP において全ての制約が充足されるならば，線形不等式制約は離散値によって充足される．

式（6.4）～（6.7）を用いて次のようなアルゴリズムにより OCSP の解を探索する．

（1）$\boldsymbol{x}$ にランダムに 0 から 1 の間の値を初期値として割り当てる．

（2）w_r，W_r を初期化する（通常，初期値としては 0 を用いる）．

（3）式（6.4）～式（6.7）に従い，力学系を更新する．

（4）もし $F_{best} > F(\boldsymbol{x})$ であり，かつ全ての制約が充足されているならば，$F(\boldsymbol{x})$ を F_{best} として更新し，現在の解を保持する．

（5）（3）～（4）を繰り返す．

上記のアルゴリズムにより，最良の目的関数の値 F_{best} を更新していく．

6.4 変数 L,W_r の役割

LPPH-OCSP は制約を充足し，かつ目的関数の値をより小さくするように解を探索する．そのため，探索では制約充足と目的関数を最小化する力をどうバランスをとるかが重要になってくる．式（6.7）のダイナミクスにおいて変数 L は LPPH-OCSP が制約を全て充足する解から抜け出すために動作する．式（6.4）～式（6.7）において L，W_r を導入しなかった場合に LPPH-OCSP がどう動作するかを調べた．15 卸売店，30 売店の WLP に L を導入していない LPPH-OCSP を用いて解くことにより得られた結果を図 6.4，6.5 に示す．図 6.4

は LPPH-OCSP の解探索を 2 次元平面状に射影したものを示している．図 6.4 が示すように LPPH-OCSP は最初に発見した 1 つの解から抜け出すことができなく，その解にとどまってしまう．図 6.5 は時間に対する LPPH-OCSP が得ることができた最良の目的関数の値をプロットしているが．LPPH-OCSP は 1 つの解から抜け出せないために，図に示されるように時間がたっても最良の目的関数の値を更新できていない．そのため全ての制約を充足する解を見つけたときには，xに対して目的関数をより小さくするような力を大きくし，その解から抜け出すことを促進することが重要である．

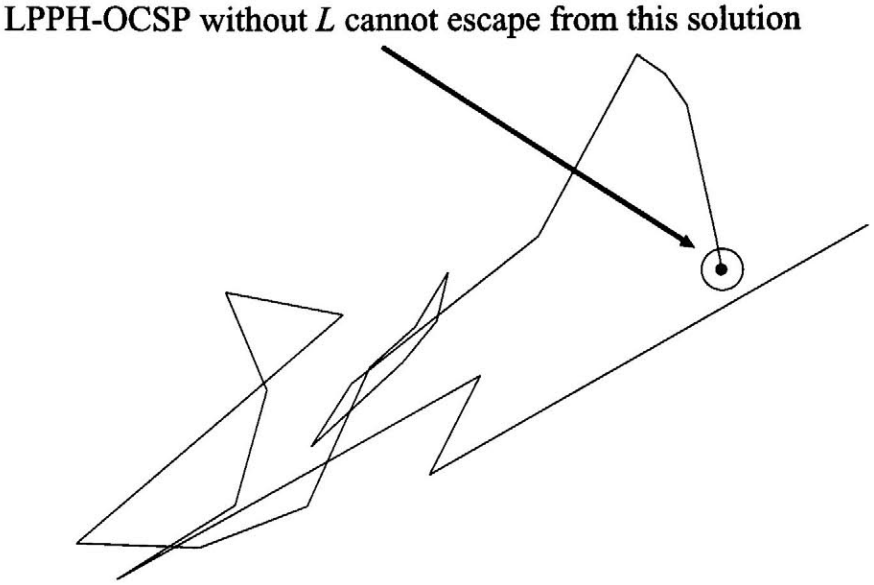

図 6.4　Lを導入していない場合の LPPH-OCSP の軌跡

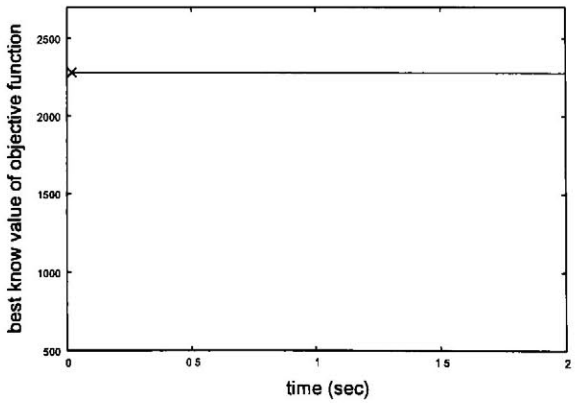

図 6.5　Lを導入していない場合の時間に対する LPPH-OCSP が得た最良の目的関数の値の変化

図 6.6，6.7 に L を導入した LPPH-OCSP(W_r は未導入) を同じ問題に適用した際の結果を示す．これらの結果から，L を導入することにより，LPPH-OCSP は 1 つの解でとまってしまうことなく探索を進めることができる．また，時間とともにより良質な解 (より目的関数の小さい解) を新たに見つけている．しかし，図 6.6 が示すように探索が進むと LPPH-OCSP はリミットサイクルに陥っていることが確認された．

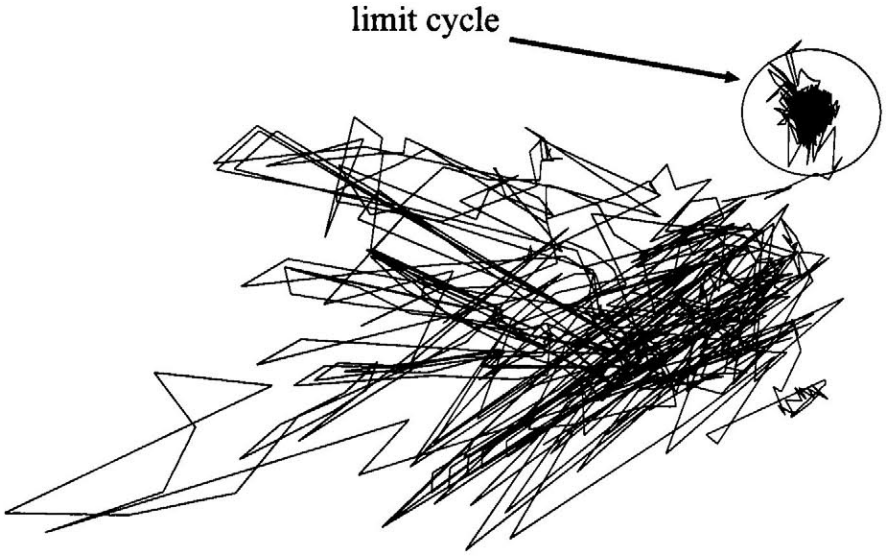

図 6.6　L を導入した場合の LPPH-OCSP の軌跡

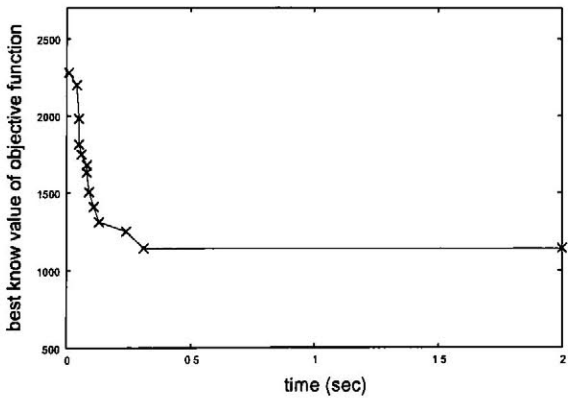

図 6.7　L を導入した場合の時間に対する LPPH-OCSP が得た最良の目的関数の値の変化

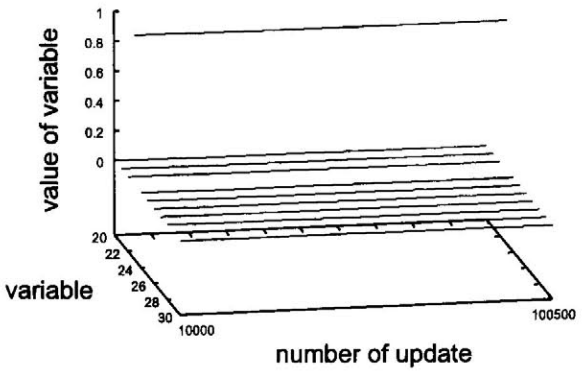

図 6.8　リミットサイクルの期間中，0，1 の値に張り付いている変数

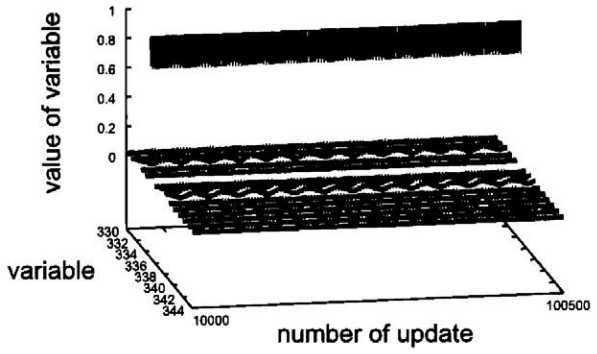

図 6.9　リミットサイクルの期間中，値の変化を繰り返す変数

LPPH-OCSP はなぜリミットサイクルに陥っているのだろうか？このようにリミットサイクルに陥った場合の LPPH-OCSP における変数の値の変化を調べたところ，図 6.8 に示すようにほとんどの変数がリミットサイクルに陥っている期間，自分の値を変えることなく 0，1 に張り付いていることが確認された．図において X 軸は力学系の更新回数，Y 軸は各変数，Z 軸は変数の値を表している．しかし，わずかな変数が図 6.9 に示されるように細かな値の変化を繰り返している

図 6.9 に現れている 15 個の変数は WLP における店 X_{23} がどの卸売り店から供給を受けるかを表現している．　ここでは“変数 X_{23} が卸売店 D_5 から品物の供給を受ける”ことを表す変数 $x_{23,5}$ が 1 に近い値をとっており，残りが 0 になっている．しかし $x_{23,5}$ は 1 に張り付いてしまうのではなく，常に $x_{23,5}$ を小さくする力が働いている．ここで $x_{23,5}$ は次に示される

ような3つの制約に出現している.

（1）ある店は少なくとも1つの卸売店の候補から供給を受ける.

（2）ある店は多くとも1つの卸売店の候補から供給を受ける.

（3）卸売店の容量制約.

上記の制約において，図6.9に示される変数の中で $x_{23,5}$ が1となり，残りの変数が0となれば制約（1），（2）は充足される．また，（3）の制約は具体的には

$$5x_{1,5} + 23x_{2,5} + 16x_{3,5} + 12x_{4,5} + 20x_{5,5} + 12x_{6,5} + 21x_{7,5} + 12x_{8,5} + 18x_{9,5} + 20x_{10,5} + 5x_{11,5}$$
$$+7x_{12,5} + 15x_{13,5} + 30x_{14,5} + 24x_{15,5} + 5x_{16,5} + 3x_{17,5} + 7x_{18,5} + 10x_{19,5} + 6x_{20,5} + 12x_{21,5}$$
$$+x_{22,5} + 30x_{23,5} + 26x_{24,5} + x_{25,5} + 16x_{26,5} + 4x_{27,5} + 10_{28,5} + 8x_{29,5} + 21x_{30,5} \leq 29x_{5,0}$$

と記述される．リミットサイクルの期間中，多くの場合において図中の $x_{23,5}$ 以外の変数は0となり，$x_{5,0}$ の値は1となっている．つまり $x_{23,5}$ の値で制約（3）の充足/非充足が決まる．この場合では $x_{23,5} = 1$ となると図6.9中の他の変数が0となってしまっても制約は充足することはない．しかし，例えば $x_{23,5} = 0.9$ ならば上記の制約は充足してしまう．しかし $x_{23,5} = 0.9$ となると制約（1），（2）が充足しなくなってしまうため，また $x_{23,5}$ を大きくしようとする力が働くためリミットサイクルに陥っている．制約（1），（2），（3）の充足のしていない度合いを表す関数 $h_r(\boldsymbol{x})$ の値の推移を図6.10に示す．なお（3）の線形不等式制約については，正規化した $h_r(\boldsymbol{x})$ の値を表示している．図6.10において縦軸は $h_r(\boldsymbol{x})$ の値，横軸は力学系の更新回数を表している．図6.10より各制約において制約の充足，非充足が交互のサイクルに陥っていることが確認できる．長期的な制約の非充足情報を扱う W_r を導入すればこのような短期間の制約の充足/非充足のサイクルから抜け出すことが可能になる．W_r を導入したLPPH-OCSPを用いて同じ問題を解いた場合の解探索の軌跡，時間に対する得ることのできた最良の目的関数の値をプロットしたものを図6.11，6.12に示す．これらの図からLPPH-CSPはリミットサイクルに陥ることなく，良質な解を時間とともに更新していることが確認できる.

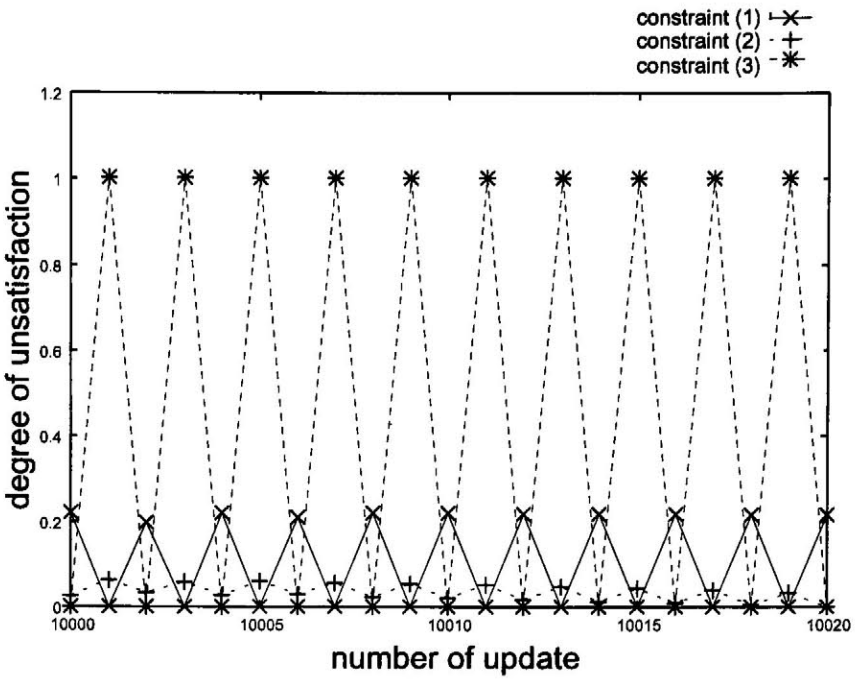

図 6.10　充足と非充足を繰り返す制約

図 6.11　L, W_r を導入した場合の LPPH-OCSP の軌跡

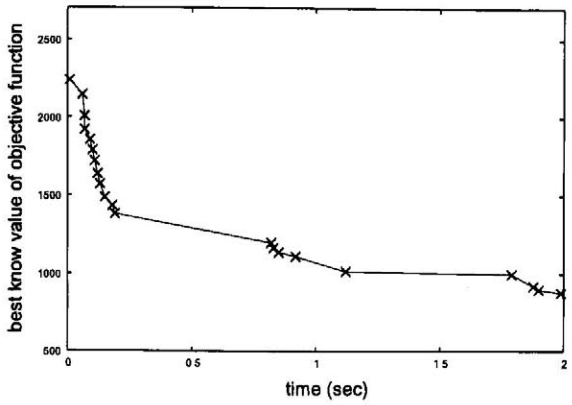

図 6.12　L, W_r を導入した場合の時間に対する LPPH-OCSP が得た最良の目的関数の値の変化

6.5　lp_solve，OPBDP との比較実験

WLP は次のように 0-1 整数計画問題として定式化することができる．

$$\text{minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}c_{ij} + \sum_{j=1}^{m} x_{j0}d_j, \tag{6.11}$$

$$\text{subject to} \quad \sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j = 1,\cdots,m, \tag{6.12}$$

$$\sum_{j=1}^{m} a_j x_{ij} \le d_j x_{j0} \qquad \forall i = 1,\cdots,n, \tag{6.13}$$

$$x_{ij} \ge 0,\ \ x_{j0} \in 0,1 \qquad \forall i = 1,\cdots,n,\ j = 1,\cdots,m. \tag{6.14}$$

ここで a_j は卸売店 j に対する販売店の要求量であり，d_j は卸売店 j の供給量の上限値を表す．LPPH-OCSP と整数計画問題の解法である lp_solve，OPBDP[19] との比較を行った．比較で用いた lp_solve は（ftp://ftp.ics.ele.tue.nl/pub/lp_solve/），OPBDP は（http://www.mpi-sb.mpg.de/units/ag2/software/opbdp/）より入手した．図 6.13~6.22 は LPPH-OCSP，lp_solve，OPBDP の比較実験結果を示している．実験では $\alpha = 0.1$，$\beta = 0.0001$，$\delta = 1$，$\mu = 30$，$\epsilon = 0.01$ を LPPH-OCSP のパラメータとして用いた．各図は時間に対してどれだけ目的関数の値が減少しているかを示しており，縦軸は得ることのできた最良の目的関数の値，横軸は CPU 時間である．図における各系列は 3 つの初期点から出発した LPPH-OCSP による結果，lp_solve，OPBDP の結果，最適解の目的関数の値をそれぞれ示している．ここで lp_solve と OPBDP は系統的探索法であり初期値依存はないので，1 つの問題に対して 1 つの結果しか示していない．用いた問題は（http://www.mpi-sb.mpg.de/units/ag1/projects/benchmarks/UflLib/）

より入手した Uncapacitated Facility Location Problem(UFLP) のベンチマーク問題を用いた．UFLP とは式（6.13）において $a_j = 1, d_j = 1$ である場合の問題である．これらの結果から LPPH-OCSP は lp_solve と OPBDP に比べ，最適解の目的関数の値に近い解を探索の早い段階で見つけていることが確認できる．また，ランダムに生成した $a_j \geq 1, d_j \geq 1$ の WLP に対しても 3 つの手法を適用した．結果を図 6.27~6.30 に示す．これらの結果からも LPPH-OCSP の有効性が確認できる．

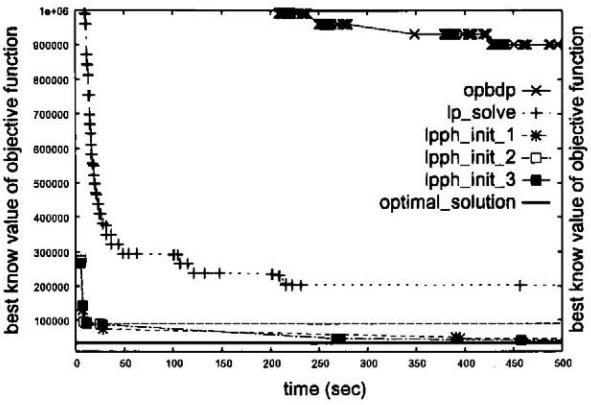

図 6.13　1032GapAS に対する比較実験結果

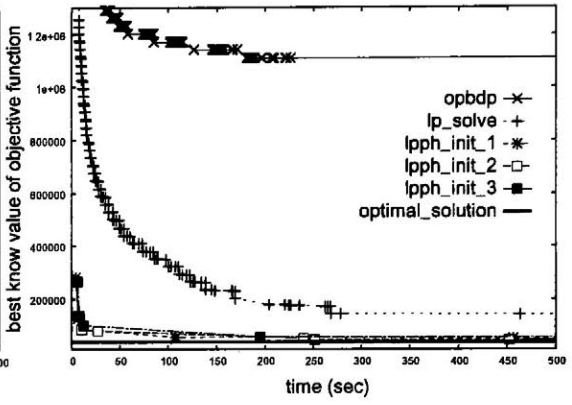

図 6.14　1532GapAS に対する比較実験結果

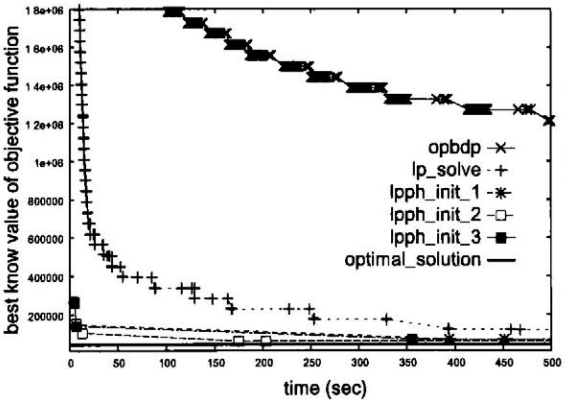

図 6.15　831GapBS に対する比較実験結果

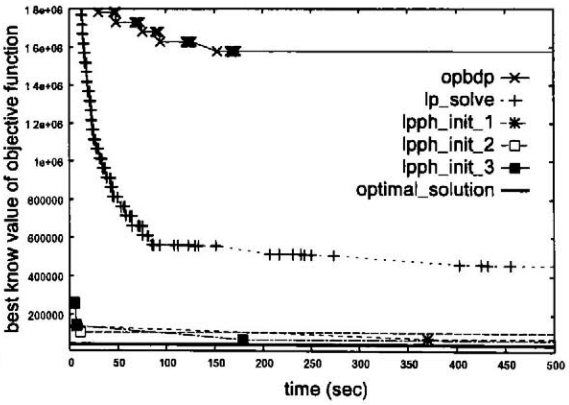

図 6.16　3231GapBS に対する比較実験結果

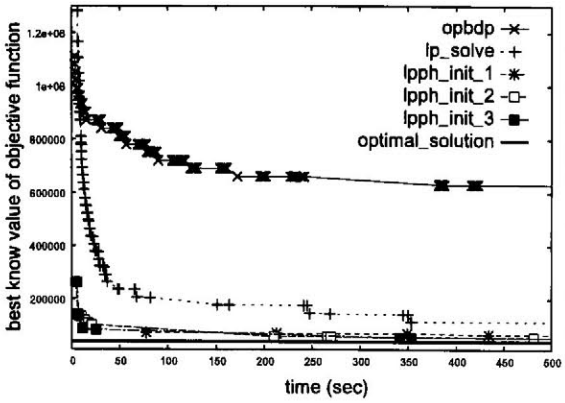

図 6.17　533GapCS に対する比較実験結果

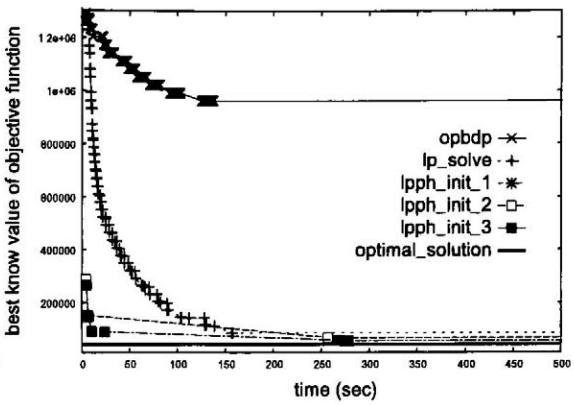

図 6.18　1033GapCS に対する比較実験結果

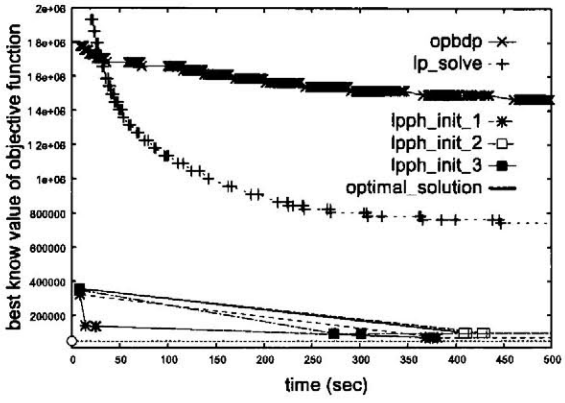

図 6.19　734PCodesS に対する比較実験結果

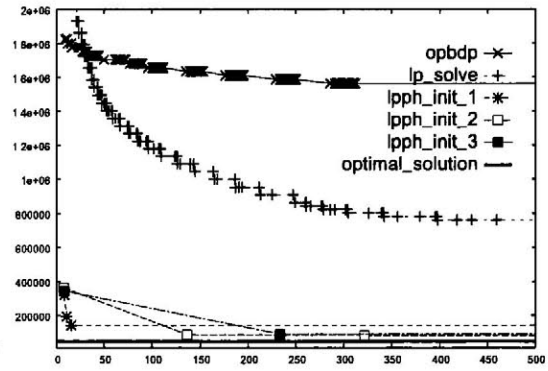

図 6.20　834PCodesS に対する比較実験結果

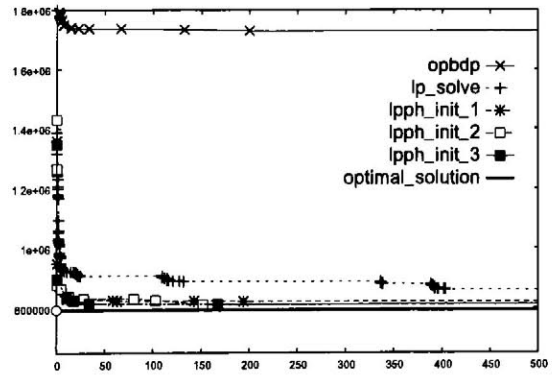

図 6.21　cap121 に対する比較実験結果

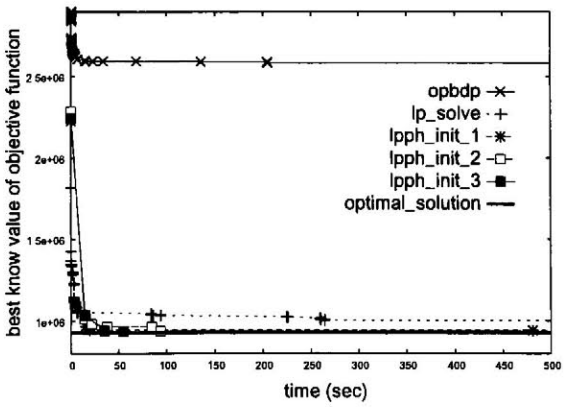

図 6.22　cap124 に対する比較実験結果

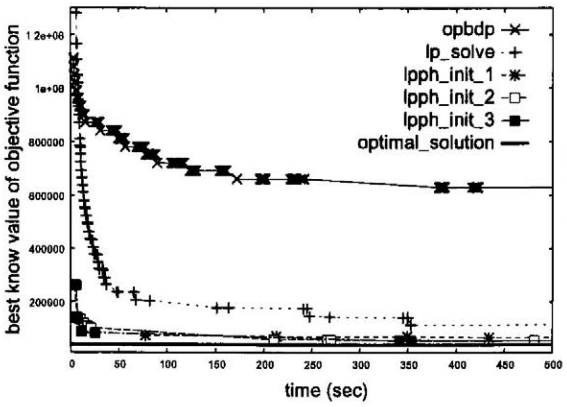

図 6.23　533GapCS に対する比較実験結果

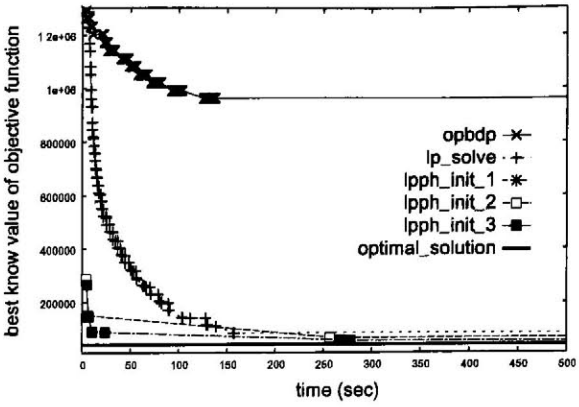

図 6.24　1033GapCS に対する比較実験結果

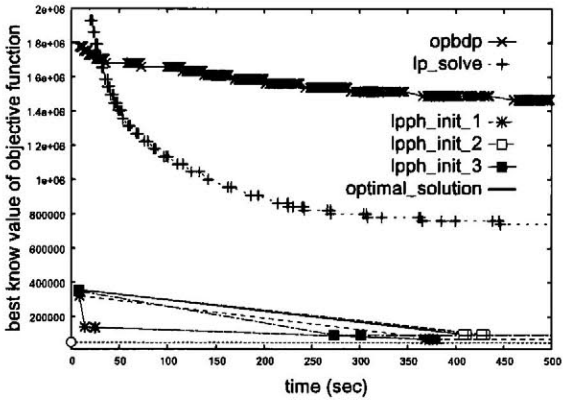

図 6.25　734PCodesS に対する比較実験結果

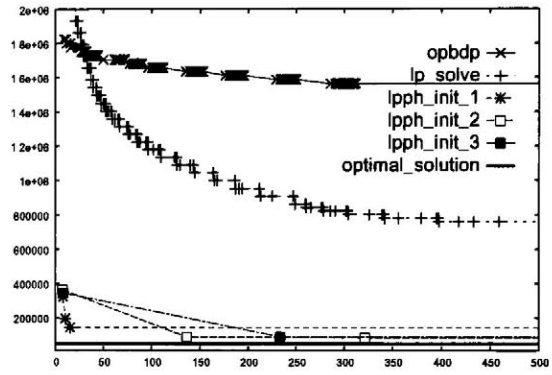

図 6.26　834PCodesS に対する比較実験結果

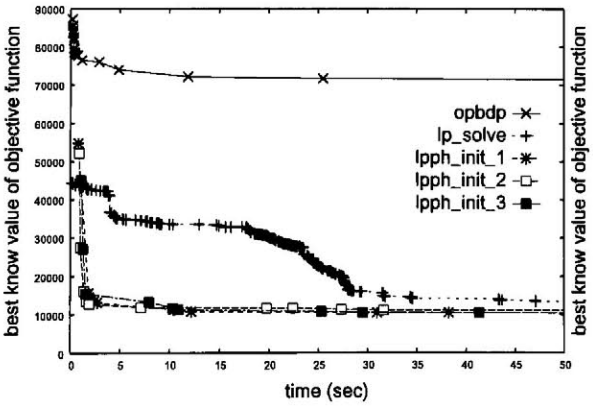

図 6.27　r1(ランダムに生成した 50 卸売店，50 売店の問題) に対する比較実験結果

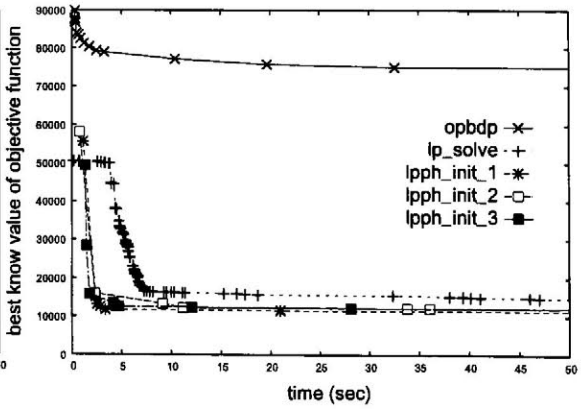

図 6.28　r2(ランダムに生成した 50 卸売店，50 売店の問題) に対する比較実験結果

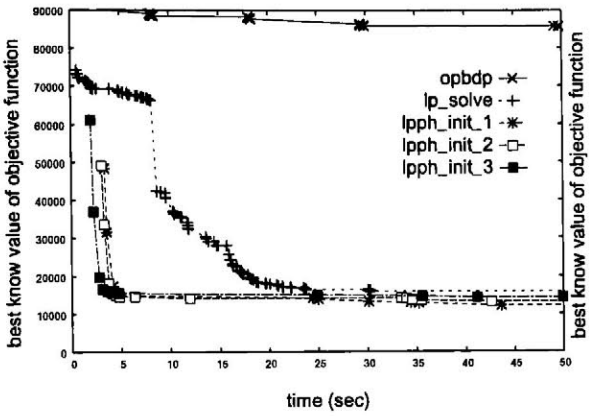

図 6.29　r3(ランダムに生成した 60 卸売店，60 売店の問題) に対する比較実験結果

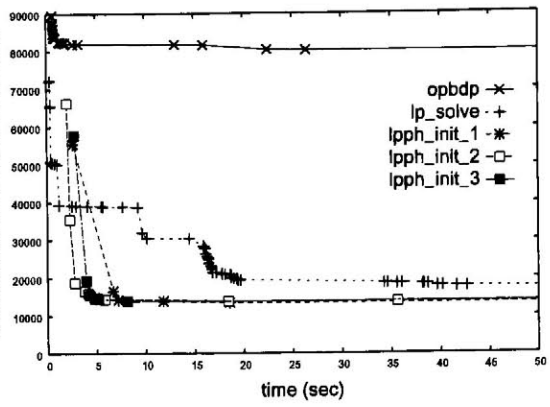

図 6.30　r4(ランダムに生成した 60 卸売店，60 売店の問題) に対する比較実験結果

第7章　結論と今後の課題

7.1　非充足関数 $h_r(x)$ と充足指示関数 $s_{rij}(x)$ の定義

SAT は CNF 論理式の節により問題の制約を表現するのに対し，CSP ではより表現能力を備えた制約表現によって問題を定式化することができる．本論文では“少なくとも n 個が真（ALT）”，“少なくとも n 個が偽（ALF）”，“多くとも n 個が真（AMT）”，“多くとも n 個が偽（AMF）”という制約表現を使った．2 項制約のみ使用する場合や SAT により表現する場合よりもよりコンパクトに問題を表現することが可能である．本論文ではまず，この制約表現を LPPH で直接取り扱うために，LPPH の拡張を行った．はじめに，SAT の節 C_r の非充足関数 $h_r(x)$ の定義を見直し，MIN 演算を用いることにより，より直感的な評価に近い定義方法を提案した．次に，このことを踏まえ，上記の 4 つの CSP 制約の非充足度関数 $h_r(x)$ と充足指示関数 $s_{rij}(x)$ の定義式として考えられる 3 つの手法（LPPH-CSP(1)，LPPH-CSP(2)，LPPH-CSP(3)）の比較を行った．LPPH-CSP(1) は非充足度関数 $h_r(x)$ を制約中の全ての変数の乗算により定義し，充足指示関数 $s_{rij}(x)$ は $h_r(x)$ の偏微分によって定義される．LPPH-CSP(2) では $h_r(x)$ は MIN/MAX 演算により，制約に現れる 1 つの変数を用いて定義される．$s_{rij}(x)$ は x_{ij} 以外の変数の中から MIN/MAX 演算によって定義される．LPPH-CSP(3) は MIN/MAX 演算により制約を複数個選びその変数を用いて $h_r(x)$ を定義し，$s_{rij}(x)$ は x_{ij} 以外の複数個の変数によって定義される．実験ではこれら 3 つの手法と SAT により問題を表現し，LPPH を用いて解いた場合について比較を行った．その結果，LPPH-CSP(2) が最もよい結果を示した．そのため，LPPH-CSP(2) を本論文の提案手法 LPPH-CSP とした．

7.2　実数値空間探索手法のよさ

GENET に代表されるように CSP の解法としては離散値をとる解法が一般的である．離散的な組合せ問題には，離散的な解空間で探索する手法に比べ，連続的な解空間で探索を進めるような解法は効率的ではないとされている．実際に幾つかの連続値をとる解法が提案され

ているが，そのどれもが効率的な解法ではない[15],[16],[30]．しかし，我々の研究室が提案したSATの解法であるLPPHは，既存の離散的なSATの解法であるGSATと比べてもより早く問題の解を見つけることができることが確認されている．CSPの解法であるLPPH-CSPは，LPPHと同様に連続値による解法であり，全ての変数の値を同時に更新しながら解を見つけだすことができる．LPPH-CSPの変数が同時に更新可能である理由は，変数が0から1の間の値をとり全ての変数の値が一斉に少しずつ変化しながら解を探索するためであることを実験により明らかにした．この特徴は従来のGENETをはじめとする離散値をとる解法では実現することができなかった．GENETでは，全ての変数の値を同時に更新すると振動現象を起こしてしまう．そのため，変数の値を1つずつ更新しなければならない．実験結果より，LPPH-CSPはGENETとほぼ同等かそれ以上の性能を示した．この比較実験は，シングルCPUのコンピュータシュミレーション結果であるため，実際の並列更新は実現していない．そのため，電子回路などのハードウェアによりLPPH-CSPを実現することができるならば，GENETのハードウェアに比べ著しいスピードアップを期待することができる．

7.3 制約に重要度を与える方法

LPPH-CSPを制約充足問題に適用する際に，問題の制約表現が探索効率に大きな影響を及ぼすことが実験によりわかった．car sequencing problemにおける容量制約の記述に冗長な制約（その制約を加えても問題の解の個数を変えることはない）を加えて記述した問題と必要最小限の制約記述で解いた問題に対してLPPH-CSPを適用し，比較実験をおこなった．その結果，LPPH-CSPは冗長な制約を加えて表現した問題の方が高速に解を見つけることができた．しかし，すべての問題に対して冗長な制約を追加した問題の方がかならず早く解を見つけ出すことができるわけではない．例えば，N-Queen問題において冗長な制約を追加した問題にLPPH-CSPを適用した場合，必要最小限の制約表現の問題に適用した場合に比べ時間がかかってしまうことを確認した．このように問題の制約表現は解法の探索効率に影響を及ぼすが，先見的な知識なしに最適な制約表現で記述することは難しい．そこで，LPPH-CSPの力学系に制約の重要度を組み込むことにより探索の効率化を図る手法を提案した．この力学系をLPPH-CSP with ICと呼ぶ．冗長な制約を追加することは、問題の付加的な情報を組み込むことと，どの制約をどれくらい重用視するかということを含んでいる．そこで提案法はLPPH-CSPのダイナミクスにその重要度を組み込んだものになっている．car

sequencing problemにおいて各制約に様々な重要度を割り振り，実験したところ，うまく重要度を割り当てるとLPPH-CSPよりも高速に解を見つけることを確認した．この重要度を問題が与えられたら問題の構造から自動的に設定する，もしくは探索とともに得られた情報をもとに動的に設定することにより，最適な制約表現の問題と同様かそれ以上の性能を得ることが今後の課題である．

7.4 LPPH-OCSPの提案とWLPへの適用

資源割当問題や輸送計画問題の実問題に目を向けると，満たすべき制約とともに何らかの目的関数が与えられる場合がある．そこで目的関数を持つCSPを定義し，その問題を解くためにLPPH-CSPの拡張を行った．この方法をLPPH-OCSPと呼ぶ．OCSPの制約としては，組合せ論的制約に加え，線形の不等式制約を考えた．線形の不等式を用いることにより，様々な実問題が表現可能になる．LPPH-OCSPでは目的関数の値を最小にする項を導入することにより，各変数は制約充足と目的関数の値の最小化のバランスをとりながら，目的関数の値のより小さい解を探索していく．LPPH-OCSPにおいて導入した変数Lは制約が全て充足しているならば，変数をより目的関数の値を小さくさせるように動作させる．実験により変数Lを導入しなければ，LPPH-OCSPは最初に見つけた解から抜け出すことができなくなってしまうことを示した．また，LPPH-OCSPにおける制約C_rの長期的な制約状況を記憶するW_rは，線形の不等式の制約充足を行う際に，リミットサイクルに陥ることを防ぐ役割を果たすことを実験により明らかにした．0-1整数計画問題の解法であるlp_solve，OPBDPとLPPH-OCSPをWLPに適用し，比較した結果，LPPH-OCSPはlp_solveやOPBDPよりもより短い時間で効率的に目的関数の小さい解を探索することを確認した．これは高速により良質な解を必要とする場合，OCSPに対してLPPH-OCSPの適用は利点があることを示す結果となった．

7.5 今後の課題

今後の課題としては，LPPH-CSP/OCSPの並列実行の利点を生かすために，電子回路を用いたLPPH-CSP/OCSPのハードウェア化が挙げられる．我々の研究室ではLPPHの電子回路を実現しているが，この技術を発展させることによりLPPH-CSP/OCSPの電子回路による実

現につながると考えている．そして制約充足問題の制約表現について，どのような冗長な制約を加えれば探索を効率化できるかを解析し，より最適な問題表現について調べるのも大事な課題であるといえる．本論文で提案した制約の重要度を考慮したLPPH-CSP with ICでは，重要度を経験的に割り当てることにより実験を行った．今後はこの重要度を自動設定する手法を開発する必要がある．重要度を設定する方法としては，与えられた問題の構造を解析し，重要度を自動的に設定する方法や，探索の過程でそれまで得られた情報から動的に重要度を設定する方法などが考えられる．本論文では，2.2節で述べた4つの組合せ論的な制約と6.1節で述べた線形不等式制約を用いて問題を表現している．しかし，組合せ最適化問題の様々な実問題を表現するのには，これらの制約のみでは十分ではないと考えられる．様々な組合せ最適化問題を調査し，それらの問題に共通してよく現れ，問題を簡略的に表現できる制約を一般化し，その制約に対する$h_r(\boldsymbol{x})$，$s_{rij}(\boldsymbol{x})$を定義していく必要があるといえる．それにより，さらに様々な実用的で大規模な組合せ最適化問題へのLPPH-CSP/LPPH-OCSPの適用が可能になる．

謝辞

本研究を進めるにあたり，多大なるご援助，御指導をして頂いた永松 正博教授に深く感謝致します．永松先生には 6 年間もの長きにわたりご指導ご鞭撻を賜りました．ありがとうございました．

本研究科・脳情報専攻　石川眞澄教授，松岡清利教授には本論文をまとめるにあたり貴重なご意見を賜りました．お礼申し上げます．

本研究科・脳情報専攻　宮本弘之助教授，森江隆教授には，COE スチューデント教育プログラムにおいてそれぞれ 2004 年 1 月～3 月，2006 年 1 月～3 月の間，研究のご指導を頂きました．ありがとうございました．

また宮本研究室，森江研究室の皆様にも大変お世話になりました．感謝致します．

なお，この学位論文の研究の一部は，生命体工学研究科　脳情報専攻　21 世紀 COE プログラム（拠点番号 J19）の推進事業として実施いたしました．関係各位ならびに関係部署に深く感謝致します．

最後に，一緒に研究活動を頑張ってきた永松研の皆様に感謝致します．

参考文献

[1] H. T. Siegelmann. *Neural Networks and Analog Computation*. Birkhauser, 1998.

[2] M. Nagamatu and T. Yanaru. On the stability of Lagrange programming neural networks for satisfiability problems of propositional calculus. *Neurocomputing*, vol.13, pp.119–133, 1995.

[3] M. Nagamatu and T. Yanaru. Lagrangian method for satisfiabiability problems of propositional calculus. In *Proceedings of the Second NewZealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, pp.20–23, 1995.

[4] M. Nagamatu and T. Yanaru. Solving SAT by Lagrange programming neural network with long and short term memories. In *Information Modelling and Knowledge Bases XI*, pp.289–301. IOS Press, 2002.

[5] M. Nagamatu, T. Nakano, N. Hamada, T. Kido, and T. Akahosi. Extensions of Lagrange programming neural network for satisfiability problem and its several variations. In *Proceedings of 9th International Conference on Neural Information Processing*, pp.1781–1785, 2002.

[6] T. Nakano and M. Nagamatu. A continuous valued neural network with a new evaluation function of degree of unsatisfaction for solving CSP. *IEICE Trans. Information and Systems*. accepted for publication.

[7] T. Nakano. A neural network for solving constraint satisfaction problem. Master's thesis, Kyushu Institute of Technologiy, 1998. in Japanese.

[8] T. Nakano and M. Nagamatu. Lagrange neural network for solving constraint satisfaction problem. *The International Journal of Innovative Computing, Information and Control*, vol.1, no.4, pp.659–671, 2005.

[9] S. Nishihara. Consistent labeling problems with applications. *IPSHJ*, no.4, pp.500–507, 1990. in Japanese.

[10] R.Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, vol.14, pp.263–313, 1980.

[11] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, vol.58, pp.161–205, 1992.

[12] C.J. Wang and E. Tsang. Solving constraint satisfaction problems using neural networks. In *Proceedings of the IEE 2nd Conference on Artificial Neural Networks*, pp.259–299, 1991.

[13] A. Davenport, E. Tsang, C. J. Wang, and K. Zhu. GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proceedings of National Conference on Artificial Intelligence*, pp.325–330, 1994.

[14] K. Nonobe and T. Ibaraki. A tabu search approach for the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research*, vol.106, pp.599–623, 1998.

[15] J. Gu. Global optimization for satisfiability (SAT) problem. *Knowledge and Data Engineering*, vol.6, no.3, pp.361–381, 1994.

[16] J. Gu, P. Purdom, J. Franco, and B. Wah. Algorithms for the satisfiability (SAT) problem: a survey. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol.35, pp.19–151, 1996.

[17] T. Nakano and M. Nagamatu. Solving CSP by Lagrangian method with importance of constraints. In *Proceedings of the International Conference on Intelligent Information Processing*, pp.255–259, 2004.

[18] T. Nakano and M. Nagamatu. Lagrange neural network for solving CSP which includes linear inequality constraints. In *Proceedings of the 15th International Conference on Artificial Neural Networks*, pp.943–948, 2005.

[19] P. Barth. A davis-putnam based enumeration algorithm for linear pseudo-boolean optimization. Technical report, MPI-I, 1995.

[20] M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In *Proceedings of the European Conference on Artificial Intelligence*, pp.290–295, 1988.

[21] H.H. Hoos. *Stochastic Local Search - Methods, Models, Applications*. PhD thesis, TU Darmstadt, 1998.

[22] M. Shuichi, I. Kazuo, and K. Yahiko. Database queries as combinatorial optimization problems. In *Proceedings of CODAS*, pp.477–483, 1996.

[23] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, vol.5, pp.394–397, 1962.

[24] D. Mitchell, B. Selman, and H. J. Levesque. Hard and easy distributions for SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp.459–465, 1992.

[25] M. Yokoo. Why adding more constraints makes a problem easier for hill-climbing algorithms: Analyzing landscapes of CSPs. In *Principles and Practice of Constraint Programming*, pp.356–370, 1997.

[26] B. Selman and H. Kautz. Domain-independent extensions gsat: Solving large structured satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp.290–295, 1993.

[27] B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pp.337–343, 1994.

[28] P. Morris. The breakout method for escaping from local minima. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp.40–45, 1993.

[29] Y. Shang and B. Wah. A discrete Lagrangian-based global-search method for solving satisfiability problems. *Journal of Global Optimization*, vol.12, no.1, pp.61–100, 1998.

[30] T. Kwok and K.A. Smith. A noisy self-organizing neural network with bifurcation dynamics for combinatorial optimization. *IEEE Transactions on Neural Networks*, vol.15, no.1, pp.84–88, 2004.

[31] J. Frank. Learning short-term weights for GSAT. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pp.384–391, 1997.

[32] F. Hutter, D. Tompkins, and H.Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, pp.233–248, 2002.

[33] Z. Wu and B. Wah. An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems. In *Proceedings of 17th National Conference on Artificial Intelligence*, pp.310–315, 2000.

[34] M. P. Scaparra and M.G. Scutella. Facilities, locations, customers: Building blocks of location models. Technical report, University of Pisa, 2001.