博士学位請求論文

単体法の反復適用による
不完全制約充足問題の解法の研究

指導教官 石塚 満 教授

東京大学大学院

工学系研究科電子情報工学専攻

斎藤 逸郎

(77114)

目 次

図 目 次

表目次

第1章　序論

1.1　研究の背景と目的

制約充足問題 (Constraint Satisfaction Problem : CSP) とは，与えられた制約を充足する値の組合せを求める問題の総称である．制約による問題の記述法は従来の推論技術における記述法と比較し，より広い範囲に応用可能である．さらに人工知能の領域における様々な問題は，いくつかの変数と制約により記述可能であるため CSP により定式化でき，実際にスケジューリング・図形処理・言語処理等の問題は制約充足問題として解く事ができる．この様な応用の広さにより CSP は人工知能の基盤技術の一つとして位置付けられており，CSP に関連した研究が多く行われ，様々な CSP の解法が提唱されている．また個々の制約に対し重みを設けることにより制約問題を最適化問題とした問題も存在する．このような重みをつけた制約問題は制約最適化問題 (Constraint optimization problem : COP) と呼ばれ，実世界に存在する問題を記述する際によく用いられている [4, 6, 22].

しかしながら実際の問題では過制約となることが多々あり，従来の制約充足問題の解法では解無しとなることが多くある．このような過制約の問題は，部分的に制約違反を認めつつ，多くの制約を満たす解を探索することにより解ける．このような問題の事を不完全制約充足問題 (Partial

CSP : PCSP) と呼ぶ．PCSP のうち，重みが付加され最適化問題としたものを不完全制約最適化問題 (Partial COP : PCOP) と呼ぶ．

多くの CSP,COP の解法では，制約に基づく局所制約伝播による整合化を行い，問題空間を狭めることにより効率化をはかっている．しかしながら PCSP,PCOP は制約違反を認めているため，局所制約伝播による整合化が困難である．そのため，特に応用範囲が広い PCOP については，効率的な解法が望まれている．

本稿では PCOP を 0–1 整数線形計画問題 (0–1 Integer Linear Problem : ILP) に帰着させ，これを解くことにより PCOP の解を得る手法について述べる．ILP の解法としては ILP の緩和問題である線形計画問題 (Linear Problem : LP) の解を単体法で求め，LP の解である実数解のうち適当なものを 0–1 に丸めることにより，問題規模を縮小させた ILP をつくることを繰り返す手法を用いており，これにより ILP の準最適解を求め，PCOP の解を求める．

1.2 論文の構成

最初に背景として 2 章で制約充足問題について，3 章で数理計画法について説明する．とくに 2 章では制約充足問題の種類とその解法について説明を行う．

4 章では PCOP から ILP への帰着法の基となった COP から ILP への帰着法についても簡単に説明を行い，PCOP から ILP への帰着法について説明を行う．また帰着による問題規模の変化についても説明を行う．

5 章では単体法の反復適用について説明を行う．特に反復適用の際に重

要になる丸め操作とその影響について説明を行う.

6章では実際にPCOPの解法を実装し，評価実験を行った結果とそれに対する考察を示す．また実際の応用例についてもここで示す.

最後に7章でまとめを行う.

第2章　制約充足問題

2.1　基本概念

2.1.1　制約充足問題の定義

制約充足問題 (CSP) を形式的に定義すると以下のようになる [22].

> n 個の変数 $x_1, \ldots, x_n$ があり，それぞれの変域を $D_1, \ldots, D_n$ とする．ただし各 D_i は有限集合でその要素を値と呼ぶ．また各二変数 $x_i, x_j (1 \leq i < j \leq n)$ について制約 $C_{i:j}$ があるものとする．ただし $C_{i:j}$ の定義域は $D_i \times D_j$ となる．この場合CSPは変数集合 ($X = \{x_1, \ldots, x_n\}$)，変域集合 ($D = \{D_1, \ldots, D_n\}$)，制約集合 ($C = \{C_{i:j} | x_i$ と x_j との間の制約. $\}$) の三つの組である (X, D, C) で定義される．

いま n 個の変数をそれぞれを $x_1, \ldots, x_n$ とし，変域を $D_1, \ldots, D_n$ とする．変域は変数に対する制約とみなす事ができる．本稿ではこの制約の事をノードに対する制約 (ノード制約) と呼び，$u \in D_i | p_i(u)$ と規定し，個々のノード制約 $p_i(u)$ をノード制約要素もしくはノードの要素と呼ぶ．また変数間の制約の事をアークに対する制約 (アーク制約) と呼び，$(u, v) \in D_i \times D_j | p_{i:j}(u, v)$ と規定し，個々のアーク制約 $p_{i:j}(u, v)$ をアーク制約要素と呼ぶ．

CSP の解とは，すべての制約 $C_{i:j}$ を満たす値の集合もしくはその要素の一つである．つまり全解が必要な場合もあれば一つで十分な場合もあり，どちらが最終解となるかは問題に依存する．

このように CSP は複数の変数について，変数の変域内で与えられた変数間の制約を満たすように，変数の値を定める問題である．

CSP の具体例を以下に示す．

$$
\begin{aligned}
&\text{変数} \quad x_1, x_2, x_3 \\
&\text{変域} \quad D_1 = \{a, b, c\}, D_2 = \{d, e, f\}, D_3 = \{g, h, i\} \\
&\text{制約} \quad C_{12} = \{(ad), (be), (cf)\} \\
&\qquad\quad C_{23} = \{(dg), (eh), (fi)\} \\
&\qquad\quad C_{13} = \{(ag), (bh), (ci)\}
\end{aligned}
$$

この問題の場合，次の三つが解となり得る．

$$(adg), (beh), (cfi)$$

上記の問題では，制約は二項制約に限定されているが，三項以上でも新たな変数を導入する事により二項で等価的に表現できる．

具体的には以下のようになる．三項制約

$$C_{123} = \{(abc), (def)\}$$

は新たな変数 $x_0(D_0 = \{\alpha, \beta\})$ を導入する事で

$$C_{01} = \{(\alpha a), (\beta d)\}$$

$$C_{02} = \{(\alpha b), (\beta e)\}$$

$$C_{03} = \{(\alpha c), (\beta f)\}$$

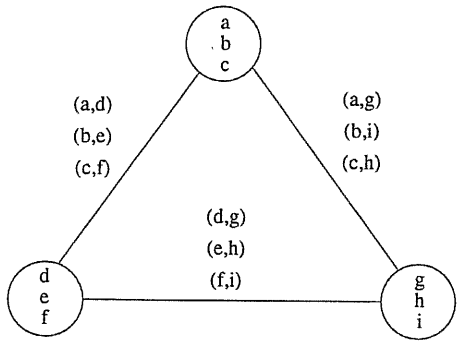

図 2.1: CSP の例

とでき，二項制約に分解できる．

通常 CSP は図 2.1 の様に，複数のノードがアークにより接続されたネットワークとして表現される．ここでノードは変数を示し，アークは変数間の制約を示す．

2.2 制約充足問題の種類

CSP そのものはすでに述べた形で定義されるが，それの拡張として様々なものが存在する．

2.2.1 制約最適化問題

制約の各要素に重みを付加した制約問題も，CSP の変形として考えられる [11, 15]．重みのついた CSP の制約の要素は，最終解の部分候補と見る事ができ，重みは部分解としての確からしさを数値化したものととらえられる．これらの部分解の重みを集積したものを充足度ととらえる事

により，制約問題を最適化問題とする事が可能である．このような重みを付けた制約問題は，制約最適化問題 (COP) と呼ばれる．COP はさらにノードの値にのみ重みがついたもの (Value COP : VCOP)，アーク制約要素にのみ重みがついたもの (Arc COP : ACOP)，ノードの値とアーク制約要素の両方に重みがついたもの (Value Arc COP : VACOP) とに分られる．本稿では各々の重みに付いて以下のように定義する．

ノードの値の重み ノードの各要素 d_{ik} において非負の値 R が与えられている時，$W(d_{ik}) = d_{ik} \times R$ で定義される値 $W(d_{ik})$.

アーク制約要素の重み アーク制約の各要素 $d_{ik:jl}$ において非負の値 R が与えられている時，$W(d_{ik:jl}) = d_{ik:jl} \times R$ で定義される値 $W(d_{ik:jl})$.

2.2.2 不完全制約充足問題

通常の CSP は，すべての制約を同時に満たす解が存在すると仮定している．そのためすべての制約を同時に満たす解が存在しない場合は，CSP の解は存在しない．

しかしながら現実の問題を CSP として記述した際に，すべての制約が同時に満たされない状態すなわち過制約 (Over-Constraint) になる場合が往々にして存在する．このような場合，ある程度制約違反を許容しつつ CSP を解くことで，大部分の制約を満たす解を得る事が可能となり，このような問題の事を不完全 CSP(PCSP) と呼ぶ [2, 7, 16].

PCSP は全ての制約を同時に満たす事が不可能であるため，PCSP の解には満たされない制約が存在する．そのため PCSP の制約は大きく二つ

に分けられる．一つは必ず満たす必要がある制約で「hard 制約」と呼ばれ，もう一つは満たされなくても良い制約で「soft 制約」と呼ばれる．

PCSP のサブクラスとして，できるだけ多くの制約が充足される解を求める問題がある．このような問題は最大 CSP(Maximal CSP : MCSP) と呼ばれている [9].

2.2.3 不完全制約最適化問題

PCSP の拡張として COP と同様に制約の各要素に重みが付いたものも考えられる．このような重みを付けた不完全性約充足問題は不完全 COP(PCOP) と呼ばれる．重みには COP の重みである，ノードの値の重みとアーク制約要素の重み以外に，各制約について制約が違反した場合の重みを定義する事で，違反する制約の重みの和を最小化する事によりなるべく多くの制約を満たす解を得る事も可能となる．最大 CSP は制約に重みが付いた PCSP のうち，制約にのみ重みがつき，かつ制約の重みが全て等しい場合とみなせるため，最大 CSP は制約に重みが付く PCSP の解法を用いる事で解ける．

本稿では制約違反した場合の重みを「アーク制約の重み」もしくは単に「制約の重み」とし，以下のように定義する．

アーク制約の重み (制約の重み) アーク制約 $d_i : d_j$ において非負の値 R が与えられている時，アーク制約が成立のときに 0 となり，アーク制約が不成立の時に R をとる値 $W(d_i : d_j)$.

2.3 制約充足問題の解法

CSP は NP 完全であり，CSP をその問題規模の多項式オーダの時間で解くアルゴリズムは存在しないと考えられる．つまり CSP は一般に組合せ探索問題である [4].

制約に基づく推論，すなわち制約条件を満足するような変数への値の割当を探索する推論プログラムは制約ソルバと呼ばれ，解構成型アルゴリズムと状態空間型アルゴリズムの 2 種類に分けられる．

2.3.1 解構成型アルゴリズム

変数への値の割当を制約に矛盾しないようにしながら徐々に拡張していく方法であり，部分解成長型アルゴリズムもしくは厳密アルゴリズムとも呼ばれる [12]．代表的なものはバックトラックによる木探索法があげられ，他には併合法等が存在する．これらはいずれも CSP の厳密解が得られるアルゴリズムである．

木探索法

通常の木探索法による方法であり，探索木の各深さレベルに変数を対応させ，分枝に値を対応させたものである．解が一つ得られる事により解決される問題では，通常バックトラックを含む縦型探索 (深さ優先探索) が用いられる．逆に最適解が必要とされる場合は横型探索 (幅優先探索) が用いられる場合も存在する．

特別な処理されていない縦型探索では，無駄なバックトラックが頻発

する事により効率が低下する事がある．無駄なバックトラックは探索木中に無駄な枝が含まれている事により発生する．無駄な枝を切り取る事により改良する方法として，フォワードチェック・ルックアヘッド・バックチェック・バックマーク・バックジャンプ等の方法が存在し，分枝刈り込みと呼ばれている．しかしながら刈り込み操作そのものに手間がかかってしまう場合があり，分枝刈り込みによる改良は必ずしも有効であるとは限らない．

併合法

木探索法によりすべての解を探索すると，最終的には探索木全体を調べ尽くす事になる．これに対して併合法はすべての解を同時に得る事ができるアルゴリズムであり，全解の探索には有効な方法である．

併合法は複数個の制約をまとめ，より多くの変数を含む一つの制約に置き換える．これを繰り返す事により，最終的に全変数を含む一つの制約にまで縮退させ，これによりすべての解を一斉に得るという方法である．

併合は関係データベースの結合演算と同じであり計算コストの高い演算である．また制約を併合する順序によって中間解の個数が大きく変動する．そのため制約の併合順序は処理効率大きな影響を与える．しかし共通変数の多い制約どうし，あるいはサイズの小さい制約関係どうしを優先的に併合するというヒューリスティックスが存在し，これを用いる事で中間解の個数を少なくする事ができ高い処理効率を得る事も可能である．

弛緩法

上記の方法はいずれも最終解を得る方法である．これらの方法を用いる前に制約どうしを突き合わせる事により無駄な要素を省き，その後に木探索法なり併合法なりを用いて解く事により，トータルとしてはより短い時間で解を得られる事が期待できる．この様な木探索法・併合法の前処理として，制約どうしを突き合わせて無駄な要素を削除していく処理の事を弛緩あるいは制約伝播という．

同時にいくつの制約を突き合わせるかによりノード整合・アーク整合・パス整合・k 整合等のレベルが存在する．しかし一般には制約ネットワークに弛緩を施しても残った要素すべてが最終解に寄与するとは限らず，最終解の目安とはなり得ない．そのため弛緩法により最終解を得る事は不可能であり，木探索法や併合法などの最終解手続きの前処理として位置付けられる．

2.3.2 状態空間型アルゴリズム

近年木探索法に代表される部分解成長型のアルゴリズムに対して，状態空間の航行という観点にたった解探索法の研究が進んでいる．これは組合せの思考錯誤によって探索を進めるのではなく，状態空間の地形を考慮しながら解の存在する場所への接近を試みる方法である．つまり全変数に解候補としての値を仮に割り当てた状態から開始し，制約違反の程度あるいは重みの総和が小さくなる状態へと進んでいく方法である．

部分解成長型アルゴリズムは厳密解が得られる事が保証されている反面，NP 完全の影響をまともに受ける．これに対して状態空間アルゴリズ

ムは厳密解で無くても次善の解が得られるので，実用的な近似解が得られればよいような問題には有効である．

状態空間探索はCSPでは制約違反の程度を，COPならば重みの総和をそれぞれの状態への適応度と考え，多次元の一価関数によりコスト面を形成する．このコスト面の特徴にあわせてよりよい解への経路を見つけ出していくものである．例としては山登り法等があげられるが，コスト面の特徴により解への接近効率が大きく異なる．そのため厳密解に十分近くて容易に到達できるようにする様々なヒューリスティックや探索法が提案されている．

リスタート法

更にこの方法では探索の候補解の選び方により，候補解のどの変数を改良しても制約違反の程度が改善されない状態に陥る事がある．この状態を局所最適と呼ぶ．この様な状態から脱出し最終解に到達するために，何らかの方法により別な候補解を設定し，再探索を行う方法である．別な候補解からの再探索も局所最適となり得るので，あるところで再探索を打ち切り解とする．打ち切る方法には，再探索を一定回数繰り返した後打ち切る方法や再探索による準最適解候補のコストの変化の仕方が一定値以下になったら打ち切る方法がある．また別な候補解を局所解からどの程度離れた点するかは地形に依存するが，様々な選択方法が提案されており，代表的なものとしてはランダムに候補解を選びだすGSATが存在する．

大域的探索法

状態空間の地形がどのようであっても多数の候補解を探索空間内に散在させておき，各々の点から並列処理的に探索を進める事により最終解に到達する可能性が高くなる．これは基本的にはリスタート法で様々な初期値から繰り返し再開するのと同じである．しかし大域的探索法では，探索が進むにつれ可能性が高い候補解周辺に重心を移すなどの適応的性質を持たせ効率の良い処理を目指している点で異なっている．

初期候補解やよりよい解候補集団の生成に乱数を用いて行う遺伝的アルゴリズムやニューラルネットを用いた方法・シミュレーテッドアニーリング法などが例としてあげられる．

2.4 不完全制約充足問題・不完全制約最適化問題の解法

PCSP・PCOPを解くアルゴリズムは以下のようなものが存在する．いずれの方法も木探索法を基にしており，全解探索・最適解探索法である．また値を代入していく変数の順序 (変数順序)，ある変数に対して代入する値の順序 (値順序) が効率に大きく影響さる [17].

2.4.1 不完全分枝限定法

不完全分枝限定法 (Partial Branch Bound : PBB) は深さ優先探索アルゴリズムであり，通常のCSPにおける木探索法を拡張したものである．基本的な動作は深さ優先探索により部分解を伸ばしていく過程において，

部分解の制約違反数が解の制約違反数の最大許容値を越えた場合にバックトラックを行う．制約違反数が解の制約違反数の最大値を越える事無く部分解を最後まで伸ばせた場合，得られた解がPCSPの解となる．制約違反数の代りに重みの総和を用いることにより，PCOPにも利用可能である．

PBBは分枝限定法の拡張であり，解の制約違反数の最大値を0とした場合，通常のCSPの木探索法と等価である．そのため解が存在する場合必ず見付かる事が保証されている．また解が得られた時点で終了する事無くすべての解を探索する事により，制約違反が最も少なくなる最適解を得る事が可能である．

2.4.2 不完全バックマーキング探索

通常のCSPにおけるバックマーキング探索は無駄な制約チェックを行わない様に木探索法を改良したものであり，以下のような手法である．

変数順序を固定した木探索において現在考慮中の変数vの各値dについて，部分解との無矛盾性を調べる．この時矛盾があれば矛盾の原因となった部分解のうち変数順序が最も小さなものを選びその順位を$mark(v,d)$として記録する．バックトラックにより既に割り当てられていた変数の値を変更する際に変数順位の最小値を$backto$として記録する．バックトラック後再び変数vの値について調べる時に$mark(v,d) < backto$ならば矛盾の原因となった部分解は変更されていなため，常に矛盾となる．そのため制約をチェックしなくてもdと現在の部分解とが矛盾すると推論でき，制約のチェックを削減できる．

不完全バックマーキング探索 (Partial Back MarKing : PBMK) はバックマーキング探索と同じ考え方により無駄な制約チェックをしなくてすむようにPBBを改良した探索法である．通常のCSPのバックマーキング探索とPBMKとの主な違いは，前者は矛盾が発生する部分解のうち変数順序が最も小さなものを記録するだけで十分なのに対して，後者は変数順序が最も小さなものと最も大きなものの値の両方を記録しておく必要がある点である．またそれに伴い，矛盾を判定する推論規則も変更する必要がある．

2.4.3 Arc Consistency Count

Arc Consistency Count(ACC) とは変数の各値毎に定義される値であり，その値より生じる制約条件違反の数を表す．前もってこのACCを知る事ができれば，PBB等のアルゴリズムにおいて現在の部分解から得られる制約条件違反数とACCとを足したものが現在の部分解を含む解の制約条件違反数となる．そのため通常より早めに枝刈りを行いバックトラックする事ができる．ACCを計算するアルゴリズムはPACCと呼ばれその計算量は c を制約の数 d を値域のサイズの最大値とすると $O(cd^2)$ である．そのため比較的大きな規模の問題ではACCを事前に計算する事により探索の効率が上がる事が期待される．

2.4.4 不完全フォワードチェッキング

フォワードチェッキング (Forward Checking : FC) とは制約伝播の手法を組み込んだ木探索法である．変数に値を代入した時に制約伝播させる

事により，まだ値を代入していない残りの変数の値域から矛盾するものを削除する事で枝刈りを行う事で探索の効率をあげる手法である．

不完全フォワードチェッキング (Partial Forward Checking : PFC) は同様に制約伝播の手法を組み込んだ PBB とみなす事ができる．PFC では変数の各値毎に Inconsistency Count(IC) と呼ばれる値を定義し，矛盾の数を記録し，制約条件違反数がの最大許容値を越える場合はバックトラックを行い，そうでない場合は部分解を伸ばす事により PBB 同様に解を探索する．

PFC では IC の計算方法等に関して自由度があるため，いくつかのバリエーションが提案されている．

第3章　数理計画法

3.1　線形計画問題

数理計画法 (mathematical programming) は変数に関する不等式の代数式で表される制約条件下で，目的関数を最小 (あるいは最大) にする変数の値を求める問題である [20]．特に，制約条件・目的関数とも線形式で表される線形計画法 (linear programming) は，単体法 (simplex method) のような高速解法が存在することから，非常に広範囲な分野で使用されている．線形でない 2 次以上の項を含む制約条件や目的関数を扱う場合は非線形計画法 (non-linear programming) となる．

変数が連続的な実数でなく離散的な整数値しかとらない場合は，整数計画法 (integer programming) となる．整数計画法で特に変数が 0, 1 の 2 値しかとらない場合は，0–1 整数計画法 (0–1 integer programming) となる．整数計画は NP 完全な問題となり，効率的な解法を得ることは難しい．従って，必ずしも最適解でなくても準最適解を効率的に見出す近似解法が重要になってくる．

数理計画法は戦時に軍の計画問題を解くことが出発点になり，オペレーションズ・リサーチ (OR) 分野で研究され，発展してきた．手法として実数の多次元空間を扱う数学的色彩の強い分野であり，離散的な記号操作を主とする人工知能の研究分野とは異なる分野と見られてきた．研究者

グループの違いもあり，現在でもそのように見ている人が多い．しかし，宣言型あるいは制約に基づく知識表現下での推論とは，制約を満たす解を見出すという点では共通であり，実は密接な関係がある．特に，知識処理で数値的に評価可能な最も良い解，最も好ましい解といった最適解を求める場合，数理計画法との関係は深い．そして，実数領域での最適解計算であるが，線形計画は単体法をはじめとして効率的解法が可能であることから，知識処理の推論の効率化においてもこの手法の利用は重要な役割を果たす．

3.1.1 線形計画問題

線形計画問題 (LP) は，実数を変域とする変数 $x_1, x_2, \ldots, x_n$ に関する制約条件と目的関数が線形の場合の計画問題である [5]．ここで制約の数を m とし，x_i のベクトル X を $X^t = (x_1, x_2, \ldots, x_n)$ とする．また a_{ij} を要素とする $m \times n$ 行列を A とし，要素数 m のベクトルを $B^t = (b_1, b_2, \ldots, b_m)$ とする．a_{ij}, b_i, c_i を定数とすると LP はつぎのように定義される．

制約条件

$$AX \leq B$$

目的関数

$$z = \sum_{i=1}^{n} c_i x_i \Rightarrow min$$

LP はその特徴として NP 完全ではなく多項式クラスの問題であることがあげられる．つまり線形計画問題に対してはその問題規模の多項式時間で解けるアルゴリズムが存在する．例えば有名な Karmarkar 内点法 (inter method) があげられる．

3.1.2 整数計画問題・0–1整数計画問題

整数計画問題 (ILP) は，整数値を変域とする変数 $x_1, x_2, \ldots, x_n$ に関する制約条件と目的関数が線形の場合の計画問題である [1]．さらに変数 $x_1, x_2, \ldots, x_n$ が $\{0, 1\}$ の値しかとらない場合は，特別な整数計画問題であるとして，0–1 整数計画問題と呼ばれる．ここで制約の数を m とし，x_i のベクトル X を $X^t = (x_1, x_2, \ldots, x_n)$ とする．また a_{ij} を要素とする $m \times n$ 行列を A とし，要素数 m のベクトルを $B^t = (b_1, b_2, \ldots, b_m)$ とする．a_{ij}, b_i, c_i を定数とすると ILP はつぎのように定義される．

制約条件

$$AX \leq b, x_i \in \{0, 1\}$$

目的関数

$$z = \sum_{i=1}^{n} c_i x_i \Rightarrow min$$

ILP と 0–1ILP は NP 完全であり，最適解が得られる解法はその問題規模の指数時間かかるものとされ，効率的な解法が重要になる．緩和法はその一つである．

緩和法は制約条件の一部を無視することにより得られる，より容易な問題を解くことにより，もとの問題を解く手法である．0–1ILP に対しては，制約条件から変数の整数条件を外し変数の変域を実数とする緩和がよく用いられる．これは連続緩和法と呼ばれ，緩和の結果得られる問題は LP となり，多項式時間で解けるようになる．0–1ILP の連続緩和問題の定義は以下のようになる．0–1ILP との違いは，すべての変数について 0 もしくは 1 である制約が，0 から 1 までの実数をとる制約に変更されている点である．

制約条件

$$AX \leq b, x_i \in [0,1]$$

目的関数

$$z = \sum_{i=1}^{n} c_i x_i \Rightarrow min$$

3.1.3 単体法

単体法は線形計画問題の代表的効率的解法であり，変数の数が数千個以下の場合は線形計画問題の最も効率的な解法として知られており，問題が解を持つ場合は有限界の反復で解が得られる．

単体法の最悪計算量は多項式オーダではないため，厳密には多項式時間アルゴリズムではない．しかしながら，多くの問題では多項式時間で解け，十分実用的なアルゴリズムである．また計算量は，近似的には制約式の数の線形オーダとなっており，実用上好ましい性質を持っている．

第4章　不完全制約最適化問題から0–1整数計画問題への帰着

4.1　制約最適化問題の高速解法

本稿で述べるPCOPの高速解法の手順は大きく二つに分けられる．一つはPCOPをILPに帰着させる前段部であり，もう一つは得られたILPについてLPに緩和し単体法を反復適用することによりILPの解を得る後段部である．

前段部の基本的な考え方については，COPの高速解法の一手法と同様であり，本稿の手法の理解の手助けにもなるので，簡単に説明を行う．

4.1.1　制約最適化問題から0–1整数計画問題への帰着

COPの高速解法にはさまざまな方法があるが，その一つにCOPをILPに帰着させ，ILPを高速解法で解く事によりCOPを高速に解く手法が存在する[26]．

この方法には，一度COPをILPに帰着させてしまえば，数多くあるILPの解法をそのまま用いる事ができる長所がある．ILPの解法もさまざ

まなものが存在するが，その一つに整数条件を緩和する事によりILPをLPとし，これを単体法などを用いて解き，LPの解を初期点として近傍探索を行うことで解を得る手法がある．この手法は，元々問題規模に対して高速に解が得られる単体法を用いているため，通常の手法より効率よく解けることが言われている．

COPをILPに帰着させる際の基本的な手法は，ノードの値の一つ一つ・アーク制約の一つ一つを0–1変数に割り当て，この0–1変数を基にして制約式と目的関数を作る方法である．

COPのうちアーク制約要素に重みがないもの，すなわちVCOPについては，アーク制約要素に対応する変数を用いずに制約式・目的関数を記述することができる．このようにすることでILPの簡素化がはかれ，解の高速化が望める．

そこでアーク制約要素に重みがある問題(ACOP,VACOP)の制約式と目的関数を「アークに基づく帰着法の制約式」と「アークに基づく帰着法の目的関数」とし，アーク制約要素に重みがない問題(VCOP)の制約式と目的関数を「ノードに基づく帰着法の制約式」と「ノードに基づく帰着法の目的関数」として説明を行う．

帰着後の0–1変数・制約式・目的関数は以下のようになる．

4.1.2 0–1変数

x_{ik} ノード i の要素 d_{ik} に対応する変数．$x_{ik}=1$ の時，ノード i が要素 d_{ik} をとる．

$y_{ik:jl}$ アーク制約要素 (d_{ik}, d_{jl}) に対応する変数．$y_{ik:jl}=1$ の時，アーク

$d_i : d_j$ はアーク制約要素 (d_{ik}, d_{jl}) により満たされる．

4.1.3 アークに基づく帰着法の制約式

アーク制約要素は一つのアークにおいて一つしか成立しない．よって一つのアークにおいてアーク制約要素の変数の和は 1 となる．

$$\sum_k \sum_l y_{ik:jl} = 1 \tag{4.1}$$

一つのアークにおいて，ノードがある値をとった時，そのノードの要素を含むいずれかのアーク制約要素が成立する．ノードがある値をとらなかった時，そのノードの要素を含むすべてのアーク制約要素は成立しない．よって一つのアークにおいて，あるノードの値に関して，ノードの値の変数とそれに関連したアーク制約要素の変数の和は等しくなる．

$$x_{ik} = \sum_l y_{ik:jl} \tag{4.2}$$

4.1.4 ノードに基づく帰着法の制約式

一つのアークのアーク制約要素のうち，一方のノードの値の変数が x_{ik} となるアーク制約要素の他方のノードの値の変数を $x_{jl'}$ とする．つまり $(x_{ik}, x_{jl'})$ を列挙したものが x_{ik} を含むアーク制約要素となるようにする．

(図 4.1 の例では x_{ik} として a をとったときは $x_{jl'}$ としては d となり，b をとったときは e に，c をとったときは f となる．逆に d をとったときは a となる．)

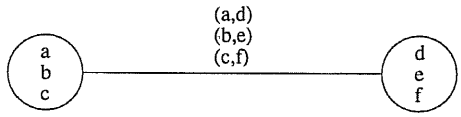

図 4.1: アークの例

一つのアークにおいて，ノードがある値をとった時，そのノードの要素を含むいずれかのアーク制約要素が成立する．ノードがある値をとらなかった時，そのノードの要素を含むすべてのアーク制約要素は成立しない．よって一つのアークにおいて，あるノードの値に関して，ノードの値の変数はそれに関連したアーク制約要素に含まれる他方のノードの値の和より小さくなる．

$$x_{ik} \leq \sum_{l'} x_{jl'} \tag{4.3}$$

あるノードに関してとり得る値は一つだけである．よってあるノードにおいてノードの値の変数の和は 1 となる．

$$\sum_{k} x_{ik} = 1 \tag{4.4}$$

4.1.5 アークに基づく帰着法の目的関数

ノードの値の変数とノードの値の重みとの積と，アーク制約要素の変数とアーク制約要素の重みの積との和が目的関数となる．

$$\min a \;=\; \sum_{i}\sum_{k} W(d_{ik})x_{ik}$$

$$+\sum_i \sum_k \sum_j \sum_l W(d_{ik} : d_{jl}) y_{ik:jl} \tag{4.5}$$

4.1.6 ノードに基づく帰着法の目的関数

ノードの値の変数とノードの値の重みとの積の和が目的関数となる．

$$\min a \;=\; \sum_i \sum_k W(d_{ik}) x_{ik} \tag{4.6}$$

4.2 基本概念

本稿で述べるPCOPの高速解法のうち，前段部ではPCOPをILPへの帰着を行う．帰着の基本的な方法は，すでに述べたCOPの高速解法と同様な手法を用いており，アーク制約に対する変数を導入することで，PCOPへの適用を行っている．

PCOPをCOPとして見ると，全ての制約を満たす事は不可能であるため，解が存在しない．そのためCOPの高速解法と同じ手法でPCOPをILPに帰着させただけでは，帰着後のILPの解は存在せず，PCOPの解は得られない．

そのためPCOPをILPに帰着させる際に常に解が存在するような帰着を行い，得られた解を基にしてPCOPの解を得る必要がある．ここで如何にPCOPを常に解がILPに帰着させるかが鍵になる．基本的な帰着方法は以下のようになる．

ノード i とノード j との間のアーク $d_i : d_j$ について，すべての組合せでアークを満たすようにアーク制約要素を追加する．すなわち，ノード i

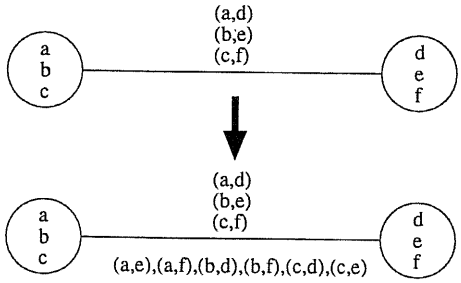

図 4.2: 基本的な考え方による帰着法

の要素を d_{ik} としノード j の要素を d_{jl} としたとき，d_{ik}, d_{jl} のすべての組合わせがアーク制約要素となるようにする．追加されたアーク制約要素の重みはアークの重み $W(d_i : d_j)$ と等しくする．

図 4.2 の例ではそれぞれの値として a, b, c と d, e, f を持つノード間のアークに，元々のアーク制約要素 $(a, d), (b, e), (c, f)$ 以外に新たなアーク制約要素 $(a, e), (a, f), (b, d), (b, f), (c, d), (c, e)$ を付け加える．

この様にする事でどのような場合でも常にアーク制約が成立し，制約が成立するので，帰着を行った ILP の解も常に存在する．PCOP の解は，元々のアーク制約が成立していれば，この解が PCOP の解となる．逆に，新たに付け加えられたアーク制約が成立していれば，その制約についてはアーク制約が成立せず，制約が満たされない事を示している．

4.3 実際の帰着法

すべての組合せを列挙すると制約が多くなり問題規模が大きくなる．そこでアーク $d_i : d_j$ に $False$ なるアーク制約要素を設ける．$False$ はノード i とノード j が如何なる値をとっても成立するアーク制約要素である．

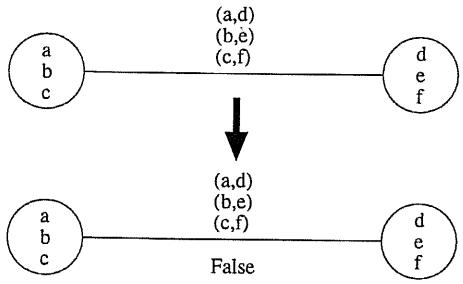

図 4.3: 実際の帰着法

図 4.3 の例ではそれぞれの値として a, b, c と d, e, f を持つノード間のアークに，元々のアーク制約要素 $(a, d), (b, e), (c, f)$ 以外に新たなアーク制約要素 $False$ を付け加える．

この様にする事で，どのような場合でも常に元々のアーク制約要素もしくは $False$ が成立するためアークを成立させる事ができるので，帰着を行った ILP の解も常に存在する．

PCOP の解は，元々のアーク制約が成立していればこれが PCOP の解となる．逆に，$False$ が成立していればその制約についてはアーク制約が成立せず，制約が満たされない事になる．

$False$ を含むアークを ILP に帰着させる際の考え方は，ノードの値の一つ一つ・アーク制約の一つ一つ・制約そのものを 0–1 変数に割り当て，この 0–1 変数を基にして制約式と目的関数を作る方法である．ILP への帰着において $False$ についてはアーク制約に含まれず，制約そのものの 0–1 変数が $False$ に相当する．以下にそれぞれの割り当て方法を詳しく説明する．

4.4 0–1整数計画問題への帰着

PCOPのうちアーク制約要素に重みがないものについては，アーク制約要素に対応する変数を用いずに制約式・目的関数を記述することができる．このようにすることでILPの簡素化がはかれ，解の高速化が望める．

そこでアーク制約要素に重みがある問題の制約式と目的関数を「アークに基づく帰着法の制約式」と「アークに基づく帰着法の目的関数」とし，アーク制約要素に重みがない問題の制約式と目的関数を「ノードに基づく帰着法の制約式」と「ノードに基づく帰着法の目的関数」として説明を行う．

変換後の0–1変数・制約式・目的関数は以下のようになる．

4.4.1 0–1変数

x_{ik} ノード i の要素 d_{ik} に対応する変数．$x_{ik}=1$ の時，ノード i が要素 d_{ik} をとる．

$y_{ik:jl}$ アーク制約要素 (d_{ik}, d_{jl}) に対応する変数．$y_{ik:jl}=1$ の時，アーク $d_i : d_j$ はアーク制約要素 (d_{ik}, d_{jl}) により満たされる．

$z_{i:j}$ アーク $d_i : d_j$ の成立・不成立に対応する変数．$z_{i:j}=1$ の時，制約は不成立となる．図4.3 $False$ に相当する．

4.4.2 アークに基づく帰着法の制約式

制約が成立する場合は，アーク制約要素は一つのアークにおいて一つしか成立しない．制約が成立しない場合は，すべてのアーク制約要素は不成立となる．よって一つのアークにおいて，アーク制約要素の変数の和と制約の変数との和は1となる．

$$\sum_k \sum_l y_{ik:jl} + z_{i:j} = 1 \tag{4.7}$$

一つのアークにおいて，ノードがある値をとった時，そのノードの要素を含むいずれかのアーク制約要素が成立もしくは制約が不成立となる．ノードがある値をとらなかった時，そのノードの要素を含むすべてのアーク制約要素は不成立もしくは制約が不成立となる．よって一つのアークにおいて，あるノードの値に関して，ノードの値の変数はそれに関連したアーク制約要素の変数の和と制約の変数との和より小さくなる．

$$x_{ik} \leq \sum_l y_{ik:jl} + z_{i:j} \tag{4.8}$$

あるノードに関してとり得る値は一つだけである．よってあるノードにおいてノードの値の変数の和は1となる．

$$\sum_k x_{ik} = 1 \tag{4.9}$$

4.4.3 ノードに基づく帰着法の制約式

一つのアークのアーク制約要素のうち，一方のノードの値の変数が x_{ik} となるアーク制約要素の他方のノードの値の変数を $x_{jl'}$ とする．つまり

$(x_{ik}, x_{jl'})$ を列挙したものがアーク制約要素となるようにする．

図 4.1 の例では x_{ik} として a をとったときは $x_{jl'}$ としては d となり，b をとったときは e に，c をとったときは f となる．逆に d をとったときは a となる．

一つのアークにおいて，ノードがある値をとった時，そのノードの要素を含むいずれかのアーク制約要素が成立もしくは制約が不成立となる．ノードがある値をとらなかった時，そのノードの要素を含むすべてのアーク制約要素は不成立もしくは制約が不成立となる．よって一つのアークにおいて，あるノードの値に関して，ノードの値の変数はそれに関連したアーク制約要素に含まれる他方のノードの値の和と制約の変数との和より小さくなる．

$$x_{ik} \leq \sum_{l'} x_{jl'} + z_{i:j} \tag{4.10}$$

あるノードに関してとり得る値は一つだけである．よってあるノードにおいてノードの値の変数の和は 1 となる．

$$\sum_{k} x_{ik} = 1 \tag{4.11}$$

4.4.4 アークに基づく帰着法の目的関数

ノードの値の変数とノードの値の重みとの積と，アーク制約の変数とアークの重みの積と，制約の変数と制約の重みの積との和が目的関数となる．

$$\begin{aligned}\min a \;=\; & \sum_i \sum_k W(d_{ik}) x_{ik} \\ & + \sum_i \sum_k \sum_j \sum_l W(d_{ik} : d_{jl}) y_{ik:jl} \\ & + \sum_i \sum_j W(d_i : d_j) z_{i:j} \end{aligned} \tag{4.12}$$

4.4.5 ノードに基づく帰着法の目的関数

ノードの値の変数とノードの値の重みとの積と，制約の変数と制約の重みの積との和が目的関数となる．

$$\begin{aligned}\min a \;=\; & \sum_i \sum_k W(d_{ik}) x_{ik} \\ & + \sum_i \sum_j W(d_i : d_j) z_{i:j} \end{aligned} \tag{4.13}$$

4.5 帰着による計算量の変化

ノードの数を n とし，一つのノードの要素の数を m とする．この時可能なアークの数の最大値 $\frac{n(n-1)}{2}$ を最大アーク数とする．また可能なアーク要素の数の最大値 m^2 を最大アーク要素数とする．アークの数を最大アーク数で割ったものをアーク密度と呼び p とし，アーク制約要素の数を最大アーク制約要素数で割ったものをアーク制約要素密度と呼び q とする．

帰着により変数の数・制約の数は変化する．それぞれを，先に定義したノード数・ノードの要素数・アーク密度・アーク制約要素密度である

n, m, p, q を用いて示すと，COP・基本的な考え方の方式・実際の帰着方法における変数の数と制約の数は表 4.1，表 4.2 の様になる．

	変数の数	制約の数
COP	$nm + m^2 q\frac{n(n-1)}{2}p$	$(1+2m)\frac{n(n-1)}{2}p$
基本的な考え方の方式	$nm + m^2\frac{n(n-1)}{2}p$	$(1+2m)\frac{n(n-1)}{2}p$
実際の帰着方法	$nm + (1+m^2q)\frac{n(n-1)}{2}p$	$n + (1+2m)\frac{n(n-1)}{2}p$

表 4.1: アークに基づく帰着による計算量の変化

	変数の数	制約の数
COP	nm	$n + 2m\frac{n(n-1)}{2}p$
基本的な考え方の方式	$nm + m^2\frac{n(n-1)}{2}p$	$(1+2m)\frac{n(n-1)}{2}p$
実際の帰着方法	$nm + \frac{n(n-1)}{2}p$	$n + 2m\frac{n(n-1)}{2}p$

表 4.2: ノードに基づく帰着による計算量の変化

変数の数・制約の数のオーダは，アークに基づく帰着では変数の数が $O(n^2m^2)$ で制約の数が $O(n^2m)$ であり，ノードの値に基づく帰着法では変数の数が $O(nm+n^2)$ で制約の数が $O(n^2m)$ となる．

アークに基づく帰着による計算量は，通常 q は 1 より小さな値をとるため，基本的な考えの方式では制約の数は変わらないものの，変数の数が大幅に増える．その点，実際の帰着方法を用いると制約の数は COP より n だけ増えるが，変数の数はそれほど変わらず，総合すると問題規模をそれほど大きくせずに，帰着できるのではないかと考えられる．

基本的な考えの方式では，アーク制約要素に対して重みを割り当てて

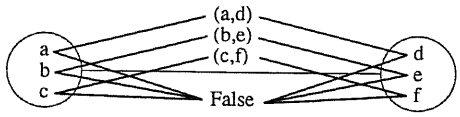

図 4.4: 変換後の制約パス

いるため，ノードに基づく帰着法が使用できずアークに基づく帰着となる．そのため元々のアーク制約要素に対して重みが割り当てられていない場合は，基本的な考え方の用いて帰着を行うと，変数・制約共に大幅に増える結果になる．一方実際の帰着法では，制約の数は変らず変数の数だけがアークの数だけ増えるため，問題規模をほとんど変えずに帰着できるものと考えられる．

これらより，本稿で述べた帰着法は，問題規模を大きく変えずに帰着が行えることが示される．特にアーク制約要素に重みがない問題に関しては，ノードの基づく帰着が行えるため，計算量をほとんど変えずに帰着可能であり，効率の良い帰着であることがわかる．

4.6 複数の制約パスの処理

以上の方法によりPCOPをILPに変換できるが，$False$を設けた事により複数の制約パスが存在し得る問題がある．

図4.4の例では，アーク制約が成立する場合としてaとdの間に(a,d)が成立する場合と$False$が成立する場合の二つの場合がある事である．

制約パスに関しては，一つのアークについてのみ影響し，他のアークに対しては関与しない．そのためあるアークに関して，成立するアーク制

約要素が $False$ と元々のアーク制約要素とが入れ替わったとしても ILP の制約式には影響がなく，目的関数の値のみが変り得る．そのため，元々のアーク制約要素の重みが $False$ より小さければ，常に元々のアーク制約要素が選ばれ，問題にはならない．

図 4.4 の例では，(a, d) と $False$ に関してはどちらの値をとっても制約式は同じであり，異なるのは目的関数の値だけになる．(a, d) の重みが $False$ の重みより小さければ目的関数を最小にするため，(a, d) が常に選ばれ，問題はない．

元々のアーク制約要素の重みが $False$ より大きい場合には正しい解が得られない．しかしながら，この状態はアークが成立した方が不成立よりコストが大きくなる状態であるので，実際の問題ではほとんど存在しないと考えられる．

仮に存在する場合は基本的な考え方と同じ方法を適用する．すなわち，複数の制約パスが問題となるアークに関して，アークに関連するノードの組合わせすべてをアーク制約要素として列挙し，それぞれに重みをつける．この場合アーク制約要素が増えるため問題規模は大きくなるものの，制約パスは一通りしか存在しないため複数のパスに関しての問題は存在しない．

4.7 最大CSPへの適用

最大CSPはもっとも多くのアークを満たす解を求める問題である．そのため制約の重みとアーク制約要素の重み・ノードの値の重みの和を目的関数とした手法では正しい解が得られない．

そこでまず最初にノードの値の重み・アーク制約の重みをすべて0とし，制約の重みをすべて1としたPCOPを解く．この解は制約の重みの和が最小，すなわち満たされないアークが最も少なくなる解である．

この解のうち，元々のアーク制約要素を満たすアークについては元々のアーク制約要素を，満たさないアークについては *False* をアーク制約要素としたPCOPを解く．この様にする事で，最大CSPをPCOP帰着させる事ができ，ILPを用いて解く事が可能となる．

4.8 soft制約・hard制約の帰着

先に述べたPCOPのILPへの帰着におけるPCOPの制約は，いずれも不成立でも構わない制約つまりsoft制約として扱っている．PCOPはsoft制約だけでなく，常に成り立つ必要がある制約であるhard制約を含む場合も存在する．

PCOPからILPへの帰着では，*False* を導入する事により，アークの不成立の表現を行っている．そのため帰着後のILPの解では *False* を含めたアーク制約のうちいずれかが必ず成立する．

そこでhard制約に関しては，*False* を導入せずそのまま元のアーク制約のみを用いる様にILPへの帰着を行う．この様にする事で，いずれかのアーク制約が満たされ，常にアークが満たされる事になり，hard制約が実現できる．

4.9 他の帰着法との比較

不完全性を持ちながら制約を解く方式としては，ここであげた方式以外に遺伝的アルゴリズムやシミュレーテッドアニーリング法を用いた方式も存在する．しかしながらこれらの手法は，すべての制約に対して不成立となる可能性が存在し，部分的に常に成り立つ必要がある制約，すなわちhard制約を設けることは困難である．

これに対してILPに帰着を行う方法では，常に成立する必要がある制約に関してはhard制約として従来の帰着法を用いて帰着を行い，不成立でも構わない制約はsoft制約として先に述べたPCOPのILPへの帰着を行うことで，解ける．

また計算量に関して，遺伝的アルゴリズムやシミュレーテッドアニーリング法が多項式オーダを保証できない．これに対してILPの近似解法には様々な方法があるがいずれも計算量は多項式オーダである．またILPの帰着法も多項式オーダであり，トータルの計算量も多項式オーダとなり，確実に解が必要とされる問題ではILPへの帰着法が有効であると考えられる．

ILPへの帰着法はこのような適用の広さの点で有用であると考えられる．

第5章　単体法の反復適用による高速解法の原理と論点

5.1　帰着により得られる0–1整数計画問題

すでに述べた方法によりPCOPをILPに帰着させることができる．ILPの解法としては，ILPの緩和問題であるLPの実数最適解を求め，実数最適解の周りの局所探索を行う手法があげられる．

しかしながら，PCOPをILPに帰着させたものは，元のPCOPが制約が強いためすべて満たせないにも関わらず，局所制約伝播による整合化が困難である特徴を持つ．そのためILPの最適解周辺でのコスト平面図5.1のように凸凹が多く局所最適が数多くあり，単純に局所探索を行うことでは近似最適解を得ることは困難であると考えられる．

またILPの各変数の解決定におよぼす影響は均等ではない．アークの成立非成立に対応する変数は一つが定まっても制約伝播は起こりにくい．アーク制約要素・ノードの要素に対応する変数は一つが定まることにより制約伝播により多くの変数が決定される．このように解への影響度は，アークの成立非成立に対応する変数比べ，アーク制約要素・ノードの要素に対応する変数の方が大きくなる．

そこでILPの緩和問題の実数最適解を求め，実数解のうち適当なものを

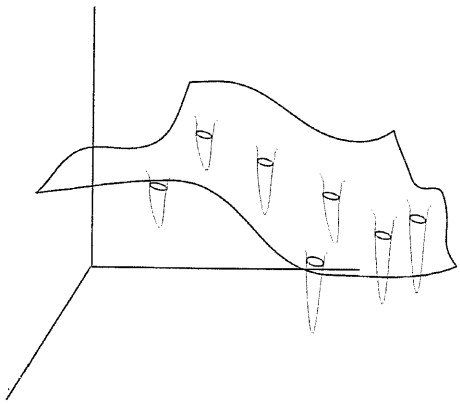

図 5.1: PCOP の解コスト面

0–1 に丸めることにより，問題規模を縮小させた ILP をつくる．縮小させた ILP も同様に実数最適解を求め，さらに縮小させた ILP をつくる．この手法を適当回数繰り返すことにより，最終的な ILP の解を得ることとした．ここで緩和問題の解法として単体法を用いることで高速化を行った．

ここで問題となるのが，得られた実数解についてどの様な方法を用いて丸めを行うかである．そこで実際の丸めの方法を説明する前に，単体法で得られた実数解が制約とどの様な関係をもつかについて示す．

5.2 実数解の収束点

PCOP より帰着した ILP の解は 0–1 解であるが，緩和問題とした単体法の解が実数解上でどの様な点に収束するかを示す．図 5.2 の PCOP について，ノードに基づく解法を用いた場合について示す．a, b を含むノードを α とし，c, d を含むノードを β とし，e, f を含むノードを γ とする．この PCOP を ILP で表すと以下のようになる．

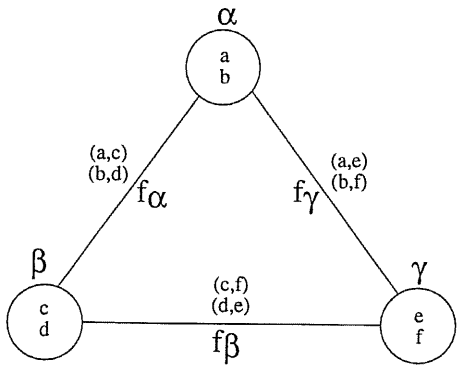

図 5.2: ノードの要素数が 2 の PCOP の例

0–1 変数　a, b, c, d, e, f に対応する 0–1 変数を $x_a, x_b, x_c, x_d, x_e, x_f$ とする．

α, β の間のアークの成立・不成立に対応する変数を f_α とし，β, γ の間のアークの成立・不成立に対応する変数を f_β とし，γ, α の間のアークの成立・不成立に対応する変数を f_γ とする．

制約式　ILP の制約式は式 (4.10) 式 (4.11) より以下のようになる．

$$x_a \leq x_c + f_\alpha, x_b \leq x_d + f_\alpha$$

$$x_b \leq x_a + f_\alpha, x_d \leq x_b + f_\alpha$$

$$x_c \leq x_f + f_\beta, x_d \leq x_e + f_\beta$$

$$x_f \leq x_c + f_\beta, x_e \leq x_d + f_\beta$$

$$x_a \leq x_e + f_\gamma, x_b \leq x_f + f_\gamma$$

$$x_e \leq x_a + f_\gamma, x_f \leq x_b + f_\gamma$$

$$x_a + x_b = 1, x_c + x_d = 1, x_e + x_f = 1$$

ここで $x_b = 1 - x_a, x_d = 1 - x_c, x_f = 1 - x_e$ とし，まとめると以下のようになる．

$$f_\alpha \geq |x_a - x_c| \tag{5.1}$$

$$f_\beta \geq |1 - x_c - x_e| \tag{5.2}$$

$$f_\gamma \geq |x_a - x_e| \tag{5.3}$$

目的関数　目的関数は以下のようになる．

$$\begin{aligned} \min a \;=\; & W(x_a)x_a + W(x_b)(1 - x_a) \\ & + W(x_c)x_c + W(x_d)(1 - x_c) \\ & + W(x_e)x_e + W(x_f)(1 - x_f) \\ & + W(f_\alpha)f_\alpha + W(f_\beta)f_\beta + W(f_\gamma)f_\gamma \end{aligned} \tag{5.4}$$

5.2.1　各変数の変域

x_a, x_c, x_e は各々独立に 0 から 1 までの値をとる．そのため x_a, x_b, x_c の変域は，図 5.3 のように x_a, x_b, x_c を軸とする空間上の立方体として表され，この中の点が制約を満たす解となる．また立方体のいずれかの頂点が PCOP の解となる．

ここで問題を簡単化するためにノードの値の重みを 0 とする．これにより目的関数を最小にするには制約の重みに関してのみ最小にすればよく，アーク制約の成立・不成立に対応する変数それぞれが最小になればよい．

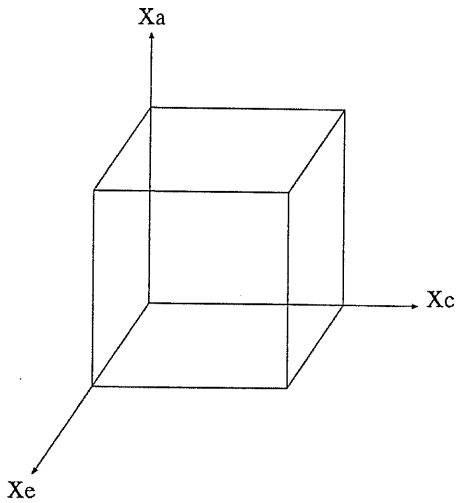

図 5.3: x_a, x_c, x_e の変域

5.2.2 各変数の収束点

ここでまず最初に f_α について示す. f_α は式 (5.1) で示されるように $f_\alpha \geq |x_a - x_c|$ となるので, $x_a = x_c$ の時に最小になり 0 となる. そのため解である立方体中の点が図 5.4 の矢印で示されている方向に向かうことで, f_α については目的関数が減少し, 矢印の先にある平面上で最小になる. f_β, f_γ についても同様にそれぞれ図 5.5 図 5.6 の矢印で示されている方向に向かうことで, 目的関数が減少し, 矢印の先にある平面上で最小になる.

次に f_α, f_β の組合わせについて示す. この場合目的関数が最小になる点は f_α, f_β のそれぞれの値が最小となる点である. そのため図 5.7 の太線で示した図 5.4 図 5.5 で示した平面同士の交線上で目的関数が最小になる. 同様に f_α, f_γ では図 5.8 の太線で f_β, f_γ では図 5.9 の太線で目的関数が最小になる.

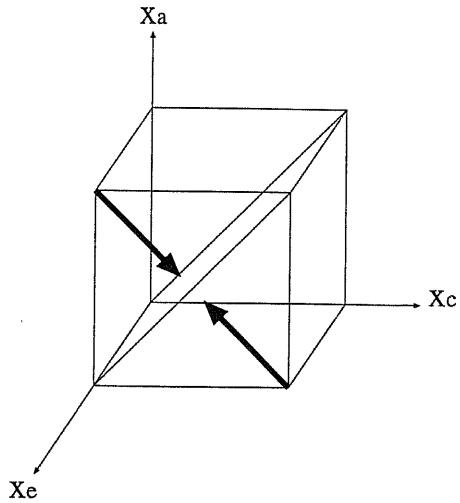

図 5.4: f_α による目的関数最小の平面

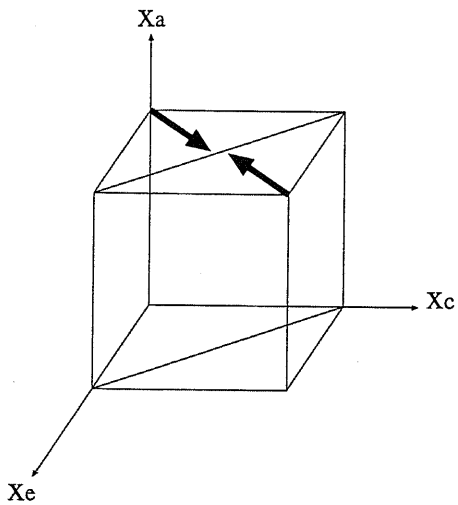

図 5.5: f_β による目的関数最小の平面

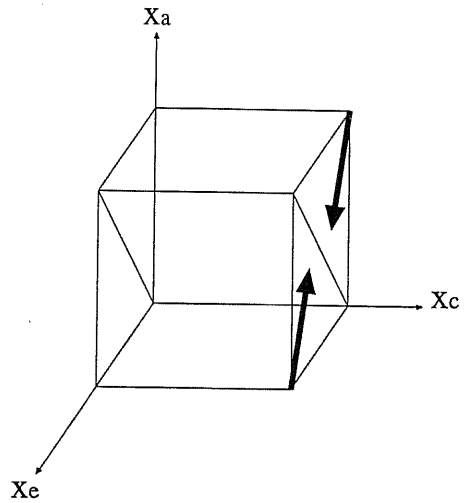

図 5.6: f_γ による目的関数最小の平面

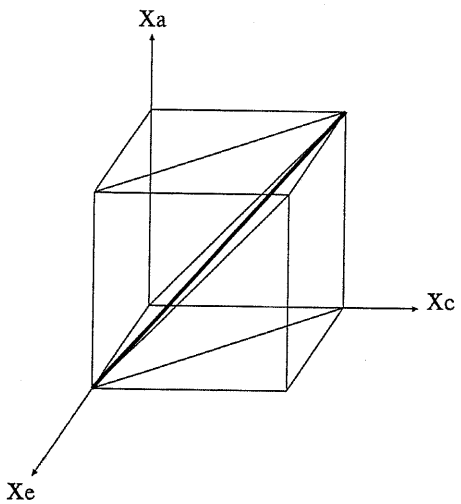

図 5.7: f_α, f_β による目的関数最小の線

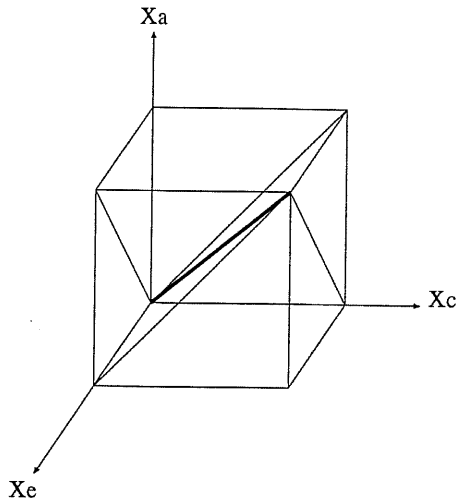

図 5.8: f_α, f_γ による目的関数最小の線

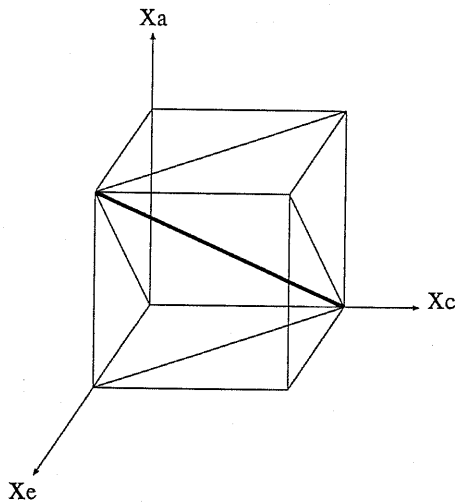

図 5.9: f_α, f_γ による目的関数最小の線

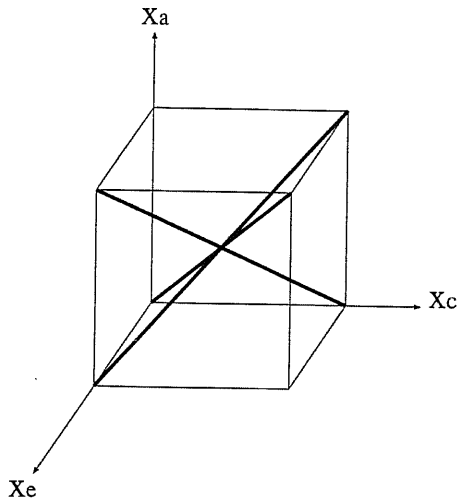

図 5.10: 図 5.2 の PCOP の帰着による LP の解

5.2.3 実数解の収束点

単体法の解は目的関数が最小になる解であるので，図 5.7 図 5.8 図 5.9 で示された交線の交点である．すなわち図 5.10 で示される交点が単体法の解となる．

5.3 単体法の解

図 5.11 の PCOP について，ノードに基づく解法を用いた場合について示す．a, b, c を含むノードを α とし，d, e, f を含むノードを β とし，g, h, i を含むノードを γ とする．この PCOP を ILP で表すと以下のようになる．

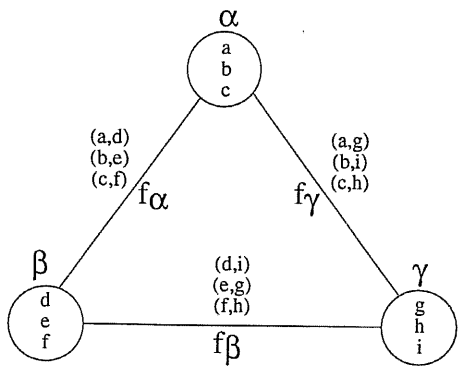

図 5.11: PCOP の例

0–1 変数　$a, b, c, d, e, f, g, h, i$ に対応する 0–1 変数を x_a, x_b, x_c, x_d, x_e, x_f, x_g, x_h, x_i とする.

α, β の間のアークの成立・不成立に対応する変数を f_α とし，β, γ の間のアークの成立・不成立に対応する変数を f_β とし，γ, α の間のアークの成立・不成立に対応する変数を f_γ とする.

制約式　ILP の制約式は式 (4.10) 式 (4.11) より以下のようになる.

$$x_a \leq x_d + f_\alpha, x_b \leq x_e + f_\alpha, x_c \leq x_f + f_\alpha$$

$$x_d \leq x_a + f_\alpha, x_e \leq x_b + f_\alpha, x_f \leq x_c + f_\alpha$$

$$x_d \leq x_i + f_\beta, x_e \leq x_g + f_\beta, x_f \leq x_h + f_\beta$$

$$x_g \leq x_e + f_\beta, x_h \leq x_f + f_\beta, x_i \leq x_d + f_\beta$$

$$x_a \leq x_g + f_\gamma, x_b \leq x_h + f_\gamma, x_c \leq x_i + f_\gamma$$

$$x_g \leq x_a + f_\gamma, x_h \leq x_b + f_\gamma, x_i \leq x_c + f_\gamma$$

$$x_a + x_b + x_c = 1, x_d + x_e + x_f = 1, x_g + x_h + x_i = 1$$

これらをまとめると以下のようになる.

$$f_\alpha \geq max(|x_a - x_d|, |x_b - x_e|, |x_c - x_f|) \tag{5.5}$$

$$f_\beta \geq max(|x_d - x_i|, |x_e - x_g|, |x_f - x_h|) \tag{5.6}$$

$$f_\gamma \geq max(|x_a - x_g|, |x_b - x_h|, |x_c - x_i|) \tag{5.7}$$

$$x_a + x_b + x_c = 1, x_d + x_e + x_f = 1, x_g + x_h + x_i = 1 \tag{5.8}$$

目的関数　目的関数は以下のようになる.

$$\begin{aligned} \min a \;=\; & W(x_a)x_a + W(x_b)x_b + W(x_c)x_c \\ & + W(x_d)x_d + W(x_e)x_e + W(x_f)x_f \\ & + W(x_g)x_g + W(x_h)x_h + W(x_i)x_i \\ & + W(f_\alpha)f_\alpha + W(f_\beta)f_\beta + W(f_\gamma)f_\gamma \end{aligned} \tag{5.9}$$

5.3.1　各変数の変域

ここで式 (5.8) より x_a, x_b, x_c の和は 1 となる. そのため x_a, x_b, x_c は図 5.12 のように a, b, c を軸とする空間上では $(a, b, c) = (0, 0, 1), (0, 1, 0), (1, 0, 0)$ で囲まれた平面上に存在する. また x_d, x_e, x_f と x_g, x_h, x_i も同様に図 5.13 図 5.14 で示される平面上に存在する.

5.3.2　ノードの値の重みによる挙動

ノードの値の重みに関して, 目的関数が最小となるには, 一つのノードの中で重みが最も小さくなるノードの値が選択されれば良い. そのた

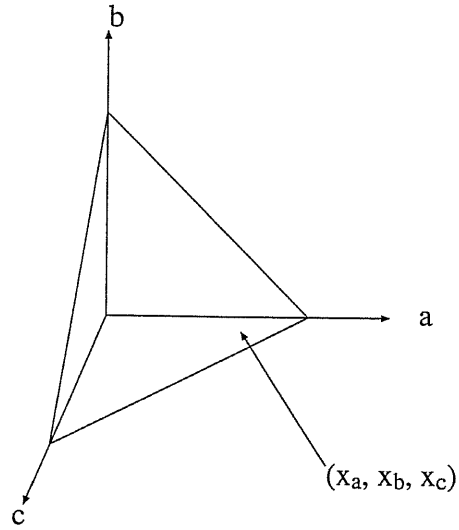

図 5.12: x_a, x_b, x_c の変域

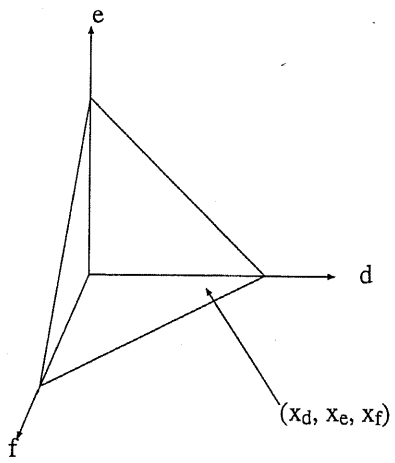

図 5.13: x_d, x_e, x_f の変域

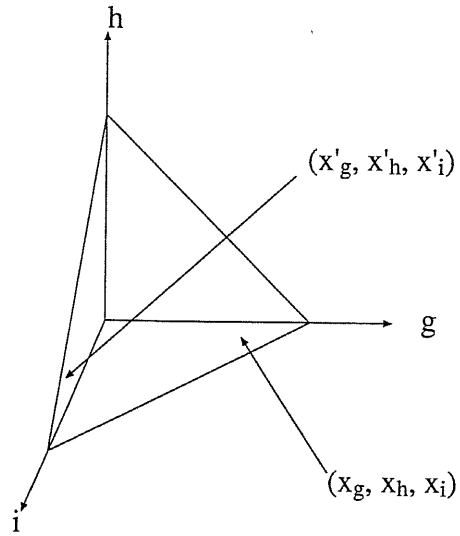

図 5.14: x_g, x_h, x_i の変域

め図 5.12 の例では，その等高線が図 5.15 で示されるようなコスト平面をなす．

よって図 5.12 図 5.13 図 5.14 では平面のいずれかの頂点に位置するときに最小となる．

5.3.3 制約の重みによる挙動

制約の重みに関して目的関数が最小となるには，アーク制約の成立・不成立に対応する変数それぞれが最小となればよい．

ここでまず最初に f_α について示す． f_α は式 (5.5) で示されるように

$$f_\alpha \geq max(|x_a - x_d|, |x_b - x_e|, |x_c - x_f|)$$

となるので， $x_a = x_d, x_b = x_e, x_c = x_f$ の時に最小となり 0 となる．つまり図 5.12 図 5.13 で解の点が同じ位置 (図 5.12 の (x_a, x_b, x_c) と図 5.13

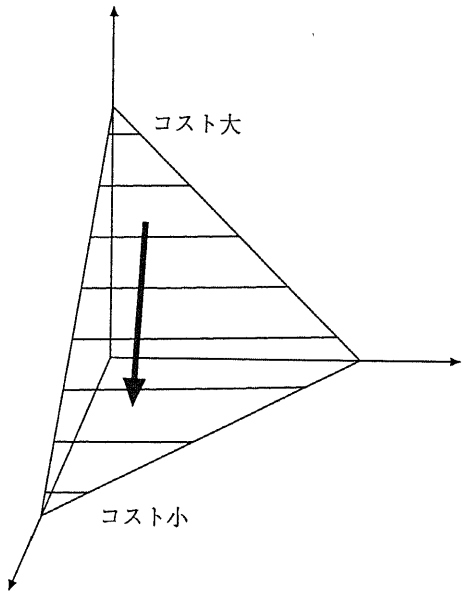

図 5.15: ノードの値の重みによるコスト平面

の (x_d, x_e, x_f) で示されている状態) にあるときに f_α が最小となる．f_γ についても同様に図 5.12 図 5.14 で解の点が同じ位置 (図 5.12 の (x_a, x_b, x_c) と図 5.14 の (x_g, x_h, x_i) で示されている状態) にあるときに f_γ が最小となる．

次に f_β についてみてみる．まず最初に図 5.14 でそれぞれの座標軸 g, h, i についてのみ入れ替えたのを図 5.16 に示す．

f_β は式 (5.6) で示されるように

$$f_\beta \geq max(|x_d - x_i|, |x_e - x_g|, |x_f - x_h|)$$

となるので， $x_d = x_i, x_e = x_g, x_f = x_h$ の時に最小の 0 となる．つまり，図 5.13 図 5.16 で解の点が同じ位置 (図 5.13 の (x_d, x_e, x_f) と図 5.16 (x'_g, x'_h, x'_i) で示されている状態) にあるときに f_β が最小となる．そのため，座標軸の配置が異なる図 5.14 では図で示された点 (x'_g, x'_h, x'_i) で最小

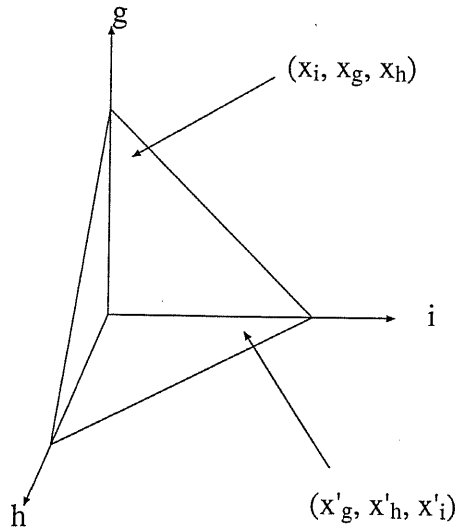

図 5.16: x_i, x_g, x_h の変域

となる.

5.3.4 単体法の解の挙動

以上の関係を一つの図にまとめると図 5.17 のようになる.

図 5.17 では x_a, x_b, x_c と x_d, x_e, x_f は近い位置に, x_a, x_b, x_c と x_g, x_h, x_i は近い位置に, x_d, x_e, x_f と x_g, x_h, x_i は離れた位置に位置するとアーク制約の成立・不成立に対応する変数が最小となる. この条件を同時に満たすことは不可能なため, アーク制約の重みにも依存するが, お互いの関係がある位置で固定される.

この時二つの点が離れれば離れるほど目的関数が大きくなる, もしくは近くなれば近くなるほど目的関数が大きくなる様子をバネを用いて示したのが図 5.17 である. 白抜きのバネが離れれば離れるほど目的関数が

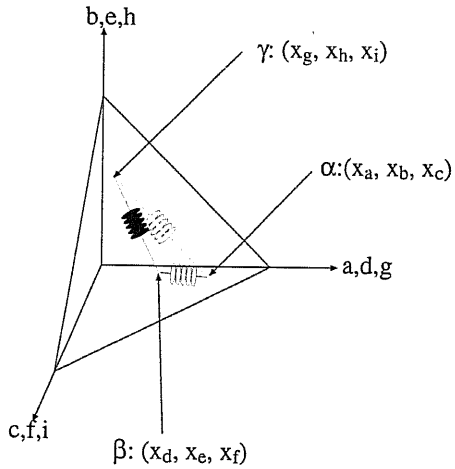

図 5.17: $x_a, x_b, x_c, x_d, x_e, x_f, x_g, x_h, x_i$ の変域

大きくなるバネを表し，黒で塗られているバネが近くなればなるほど目的関数が大きくなるバネを表している．それぞれのバネの強さはアーク制約の重みに依存する．

ノードの値の重みも含めて目的関数が最小となる状態は，それぞれのバネの状態が安定した状態であると言え，逆にLPを単体法を用いて解くことにより安定した状態を求めているととらえられる．また目的関数の値はその状態の安定度を示しているともとらえられる．

5.4 丸め操作の意味

単体法で得られた解のうち適当なものを0–1に置換えることにより丸めを行う．ここで丸めの対象となる変数を，ノードの値に関する変数に限定して考える．この場合丸めの操作は，先にあげた図5.12を用いて示すと，x_a, x_b, x_c の値を $(0,0,1), (0,1,0), (1,0,0)$ のいずれかに固定するこ

とに他ならない．この作業は図 5.12 でみると，単体法で得られた解が平面上のいずれかの点を示しているのを，平面の頂点へ移動するとらえることができる．

5.5 単体法の反復適用の原理

ある値について丸めを行うことにより，目的関数や制約の状態が変化する．この変化は，一部は制約伝播により一意に決定される変化であるが，多くの変数に関しては，変数の変域が狭まる程度にとどまり，一意に決定されることは少ない．

この様子を図 5.17 を例として示す．ノード α について丸めを行い $(x_a, x_b, x_c) = (1, 0, 0)$ にしたとする．このようにすると x_d, x_e, x_f と x_g, x_h, x_i はともに x_a, x_b, x_c に引かれるため，丸め先である $(1, 0, 0)$ に近づこうとするが，x_d, x_e, x_f と x_g, x_h, x_i が反発しあうため，そのまま $(1, 0, 0)$ とはならず，ある点で安定する．

そこで α について丸めを行い固定したことを示す $x_a = 1$ という制約式を元の ILP に付け加え，その緩和問題である LP を単体法を用いて解く．このようにすることで，α については丸めを行い固定した状態での最適解が得られる．

これは図 5.17 で示すと α の点を $(1, 0, 0)$ に固定することで，バネが引っ張られたり縮められたりされ，その結果として最安定点が移動する．α の点を移動し固定したことを示す制約を付け加えた ILP を，その緩和問題である LP を解くことにより，再度最安定点が求められ，他のノードである β, γ についてはそれぞれ安定な場所に移動する．

図 5.11 の例は簡単なため，一つのノードを丸めることで移動し固定しても他のノードへの影響は少ない．しかしながら多くのノードが複雑に絡み合っている PCOP では，一つのノードを丸めて移動した場合，他のノードがどの様に変化するかは定かではない．先にあげたバネのモデルでは単純な構造では単に平行移動するだけですむが，複雑に絡み合っていると，ゴム紐におけるクンキングの様に相転移が生じ，突如ノードの配置が変ることがある．

丸めを行い固定した状態で再度単体法の計算を行うことは，その固定した状態での最適解を得ることに他ならないので，相転移にも対応でき，上記のような問題は発生しない．このように相転移のような大きな変化が生じても常に最適解が得られるため，単体法の反復適用は局所解に捕らわれにくくなり，準最適解が得られるようになる．

5.6 丸めを行う値の選択

このように徐々に丸めを行いながら単体法を用いて最適解を算出することで，局所最適を避けながら準最適解が得られる．ここで問題となるのが，どのノードについて丸めを行うかという点と，どの値に丸めるかである．

5.6.1 丸め先の選択

まず後者について考える．どの値に丸めるかは，丸めによる制約の状態への影響が最も少ない点について丸めを行うのが良いと考えられる．制

約の状態は図 5.17 の様になっていると考えられ，この状態で制約の状態への影響が最も少ないと考えられるのは，丸めを行う先である平面の頂点群 $(0,0,1),(0,1,0),(1,0,0)$ のうち最も近い点へ丸めを行うことである．図 5.17 の例では α,β については $(a,b,c)=(1,0,0),(d,e,f)=(1,0,0)$ の点が γ については $(g,h,i)=(0,1,0)$ の点が最も良い丸め先となる．

ノードの各要素の値に対応する変数を x_{ik} とし，単体法の解のうち最も大きな値をもつ変数を $x_{ik_{max}}$ とする．この時ノード i について丸めを行う場合，丸め先は $x_{ik_{max}}$ を 1 とし，それ以外を 0 とする点が最も良いと考えられる．

5.6.2 丸めるノードの選択

前者については，一度に幾つのノードについて丸めを行うかという問題も含まれる．一度に丸めを行うノードの数を 1 に限定すると，徐々に解に近づくため，より良い解が得られると考えられる．この場合，単体法の反復適用ではバックトラックを行わないため，ノードの数と同じ回数だけ単体法を解けば必ず解が得られる．しかしながら，このように一度に丸めを行うノードの数を限定すると，単体法に要する時間が短いとはいえ，無駄に時間がかかることになる．

一方多くの値について同時に丸めを行うことで，単体法の適用回数が減り，全体の計算時間を短くすることができる．しかしながら，このように一度に多くの値を丸めると，丸めの課程で相転移が発生し最適解が大きく変化したときに対応できず，局所最適に陥やすくなる．

そのため，どの様な値をとるノードについて一度に丸めを行うかが，解

の速度と解の質に影響し，重要な問題となる．

どのノードについて丸めを行うかは，丸めによる制約の状態への影響が少ないノードに対して行うのが良いと考えられる．制約の状態は図 5.17 の様になっていると考えられるので，制約の状態に対して影響が少ないと考えられるノードは，丸めを行う先である平面の頂点群 $(0,0,1)$, $(0,1,0)$, $(1,0,0)$ のいずれかに対して最も近い点に実数解が存在するノードである．図 5.17 の例では，α は他の β,γ に比べて頂点に近くに存在し，丸めによる制約の状態への影響は，α は他の点よりも小さくなると考えられるので，α について丸めることにより良い解が得られると考えられる．

この事を定式化すると以下のようになる．

ノードの各要素の値に対応する変数を x_{ik} とし，単体法の解のうち最も大きな値をもつ変数を $x_{ik_{max}}$ とする．この時ノード i について，実数解と丸め先との距離 ND_i は以下の式で求まる．

$$ND_i = \sqrt{\sum_{k \neq k_{max}} x_{ik}^2 + (1 - x_{ik_{max}})^2} \tag{5.10}$$

この ND_i を各ノードについて求め，小さいものに対して丸めを行うのが良いと考えられる．

5.7 丸めの手法

そこで以下のような丸めの手法を用いることとした．

1. ノードの値に対応する変数についてのみ丸めを行う．
2. 各ノードに関して ND_i を求める．

3. $ND_i = 0$ となったノードについては，実数最適解が整数解であるので，実数最適解を用いる．

4. $0 < ND_i < 0.5$ となったノードについては，先に述べた手法により丸めを行う．

5. 4. の処理が行われなかったとき，ND_i が最も小さな値となるノードに対して，先に述べた手法により丸めを行う．

6. 5. の処理で，ND_i が最も小さな値となるノードが複数存在した場合は，それぞれのノードについて，そのノードについて丸めを行ったILPの緩和問題であるLPについて単体法を用いて解きその目的関数値を求め，目的関数値が最小となった時に丸めを行ったノードを，反復過程の丸めとして用いて，丸めを行う．目的関数値が同じ値をとる丸めが複数存在したときはいずれかを選択する．

1. は，アーク制約要素に対応する変数やアーク制約に対応する変数は，ノードの値の変数によって一意に決定されるため，ノードの値についてのみ決定すればCOPの解が決定されるためである．

3. は，すでに解が得られているノードに関してはその解を用いることを示している．

4. は，実数解と実数解を丸めた値との距離が一定値 (以下この値のことを「丸めの境界値」と呼ぶ) 以下のものについて一度に複数個の解について同時に丸めることを示している．これにより単体法の反復適用の終盤において多くの値を同時に決定することが可能となり，無駄な単体法の計算を省くことが可能となる．丸めの境界値として大きな値をとると局所最適に陥りやすくなり，逆に小さな値をとると無駄な単体法の計算が

行われる．ここで，丸めの境界値を幾つにするかが問題となる．現在 0.5 を用いているが，これは丸めの境界値を変えつつ PCOP を解き，解と計算時間に基づいて決定した．この結果については 6.3 節で述べる．

6. は，実数解を丸める際に，ND_i に基づいて判別できないときには，実際に丸めた結果を求めることで，判別を行っていることを示している．これにより，丸めによる局所最適が避けやすくなっている．

5.8 反復適用による局所最適回避

すでに述べたように丸めを行う際に，丸めと丸めの間で単体法を再計算することにより，妥当な丸めを行い局所最適に陥るのを避けている．実際に反復適用を行う事でいかに局所最適を回避しているかについて示す．

状況を簡素化するため，PCOP のノードのうち 2 ノードについてのみ考える．2 ノードを x, y とし，x, y により構成されるコスト平面を図 5.18 に示す．最適解である単体法の解は図 5.18 のうち，A で示される点である．

このコスト平面は x, y の変域を実数としたときに得られるコスト平面であり，実際の解のコスト平面とは異なる．ここで，ノードの変域を整数と仮定して，実際の解のコスト平面を図に示すと，図 5.19 のようになる．

5.8.1 ノードの丸め処理

丸め操作はノードの変域を実数から実際の変域に置換えることである．x について丸めを行ったコスト平面は図 5.20 に，y について丸めを行っ

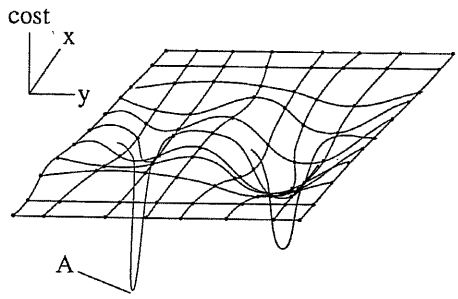

図 5.18: 実数領域でのコスト平面

図 5.19: 整数領域でのコスト平面

たコスト平面は図 5.21 に，示すようになる．

丸めを行うノードは，ノードの実数解と丸め先との距離が最も小さなノードが選択される．x について丸めを行ったときの丸め先は，図 5.20 の線群上の点のうち最適解に最も近い点であり，y について丸めを行ったときの丸め先は，図 5.21 の線群上の点のうち最適解に最も近い点である．

そのため丸めは，各ノードについて丸めを行うことで得られる線群のうち，最適解に最も近い点をもつ線群になるノードが，丸めを行うノードとして選択される．

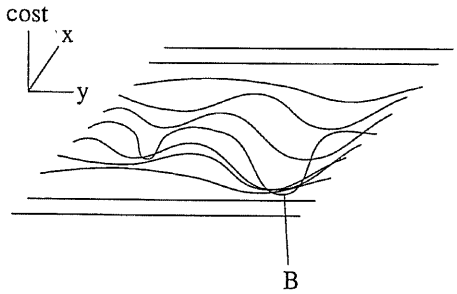

図 5.20: xについて丸めを行ったコスト平面

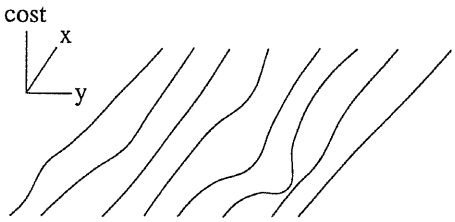

図 5.21: yについて丸めを行ったコスト平面

図 5.18 の例では図 5.21 で示される線群に比べ，図 5.20 で示される線群の方が最適解である A に近い線をもっているため，丸めを行うノードとしては x が選択され，丸め後のコスト平面は図 5.20 となる．

5.8.2 単体法の再適用

丸めを行った後，再度単体法を用いて最適解を求めることは，丸め後のコスト平面 (線群) 図 5.20 のうちコストが最小となる点を求めることである．図 5.20 のうちコストが最小となる点は B で示される点となり，図 5.18 でコストが最小となる点である A とは大きく異なる点である．

これは A での最適値が鋭い最適値であるためである．このような点は

変域を実数解としたときには最適解となるが，実際の解である整数解では実数解とは距離は近いにも関わらずコストは大幅に異なり，最適解からはずれ局所最適となる．

単純に丸めを行うだけではこのような局所最適に陥り，準最適解を求めることは困難となる．また当然のことながら，このような点に関しては実数最適解を丸めることにより得られた初期点からの山登り法などを用いても改善されない．

一方丸めを行った後，再度単体法を用いて最適解を求めることで，丸めを行った状態での最適解が得られ，このような局所最適を逃れることができ，準最適解が得られる．

5.9 山登り法による解の改善

このように他の状態空間型アルゴリズムと比較した場合，単体法の反復適用の最大のメリットは，一度局所最適付近に近づいても，単体法の再計算により逃れることが可能である点である．

単体法の反復適用ではバックトラックを行わないため，一度丸めを行ったノードは変更されることはない．そのため誤った丸めを行うと最適解にたどり着けなくなる．しかしながら再計算の結果得られる値は丸めを行った状態での最適解であるため，その近傍には準最適解が存在するものと考えられる．

そこで反復適用毎に得られた実数解について丸めを行った点を初期点とし，その点から山登り法により近傍探索を行う．実数解の丸めは先に述べたのと同じ手法により行う．これにより複数の準最適解が得られ，そ

のうち最も良い解を選択することにより，より良い解が得られる．また山登り法ではすべての変数に関して0–1が書き換えられる可能性があるため，一度丸めを行い決定された変数に関しても書き換えられる可能性がある．そのため丸めの過程で誤った変数に関しても山登りの過程でコストが小さくなるように書き換えられ得るため，誤った丸めによる局所最適からも脱出可能となる．

第6章　不完全制約充足問題の単体法反復適用による解法システムとその評価

6.1　RS解法システムの実装

先に述べた手法によりPCOPを解く手法(以下RS法(Repeat use of Simplex method)と呼ぶ)の処理系として，RS解法システムの実装を行った．処理の流れを図6.1に示す．入力としてはノード数・ノードの要素数・アーク・アーク制約要素およびそれぞれの重みが列挙されたファイルである．処理系はパラメータとして丸めの境界値をとり，丸めの境界値を変えたときの解コスト・計算時間の変化を見る実験の結果より，通常は0.5を用いている．出力は各ノードの要素とそのときの解コストであり，ファイルに出力される．また生成したLP・LPの実数最適解・実数最適解を丸めた結果については，計算の中間結果としてファイルに保存できるようになっている．LPの実数最適解を求める単体法の計算は`lp_solve`[28]を用い，山登り法は`C`を用いて記述し，それ以外の設定ファイルの読み込み・単体法の呼び出し・丸めの処理に関しては`ruby`[29]を用いて記述した．

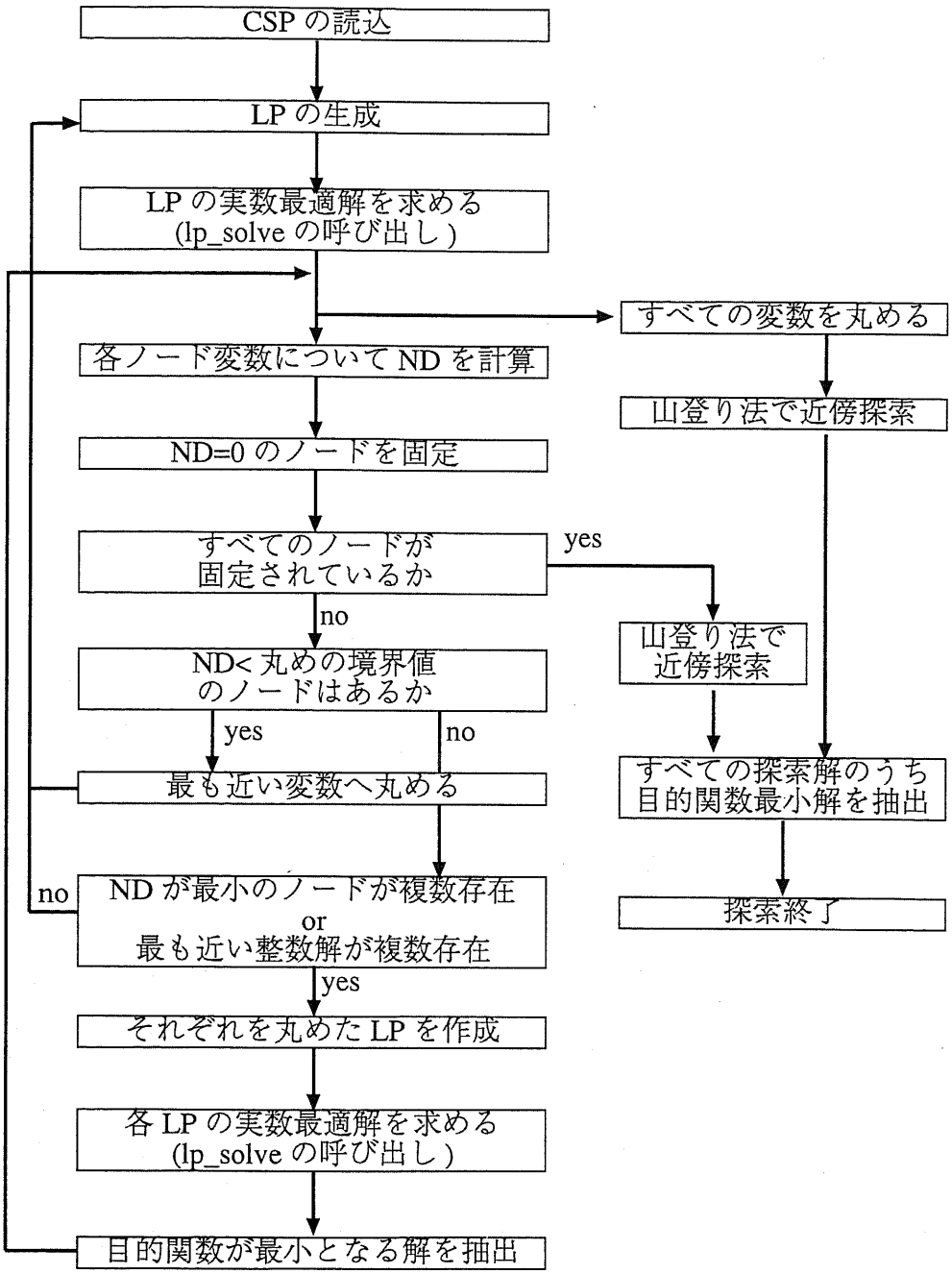

図 6.1: 実装を行った処理系の流れ

6.2 評価問題の生成

CSPでは解法によって得意な問題・不得意な問題が存在するため，解法の評価は，様々な問題を取り集めた標準問題を解かせる事で行われる．しかしながらPCOPでは標準問題に相当するものが無く，評価の問題を作るところからはじめる必要がある．ここではPCOPの問題として広く用いられている，グラフの塗り分け問題を使用し，評価問題の自動生成を行い解法の評価を行った．

6.2.1 グラフの塗りわけ問題

グラフの塗り分け問題は，4色問題を拡張したものである．4色問題とは図6.2の様に幾つかに分割された平面を隣同士が異なる色になるように塗り分ける問題であり，4色あればどの様に分割されていても塗り分けられることが示されている．図6.2をCSPで記述すると，図6.3の様になる．ここで，分割された平面がノードで表され，その色がノードの値となり，平面同士が隣り合っていることがアークにより表現されている．ノードの値・アーク制約は各ノード・各アークで全て同じであり，各ノードは a, b, c, d のいずれかからなり，アーク制約は $(a, b), (a, c), (a, d), (b, c), (b, d), (c, d)$ となる．

平面を分割しただけのものを一般のグラフに拡張したものがグラフの色分け問題である．グラフの色分け問題は，アークで接続されたノード同士が同じ値をとらないように，ノードの値を設定する問題である．ノード同士が同じ値にならないようにするために必要とされるノードの要素数はアークの付け方により変化し，必要最低限のノードの要素数が n と

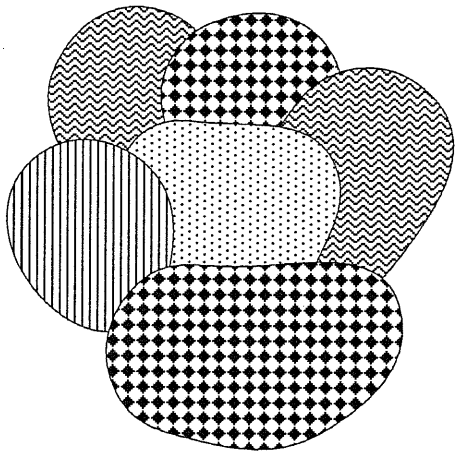

図 6.2: 4 色問題の例

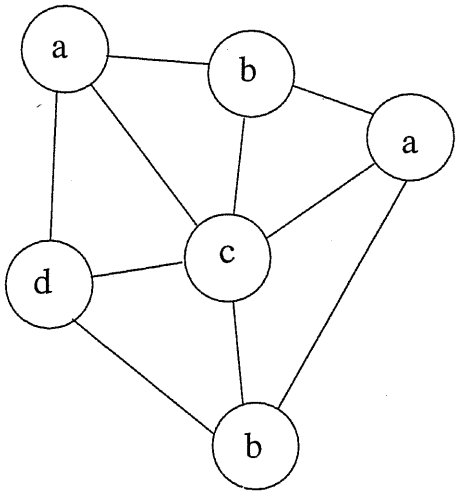

図 6.3: 図 6.2 の 4 色問題の CSP による記述

なる問題をn色問題と呼ぶ.

ここで，n色問題について，ノードの要素数を n より小さな値に設定する．この場合アークで接続されたノード同士が同じ値をもつアークが必ず発生する．このアークはn色問題の制約を満たさないアークであり，このようなn色問題は常に不成立となる制約を含みPCSPとして表現される．さらに，ノードの値やアーク制約に適当に重みをつけることによりPCOPが生成できる．

6.2.2 不完全制約最適化問題の自動生成

ノードの数を n とし，一つのノードの要素数を m とする．またアーク密度を p とし，アーク制約要素密度を q とする．実装を行ったPCOPの自動生成は，四つのパラメータ n, m, p, q を引数として生成を行う．アークの設定・アーク制約要素の設定はアーク密度・アーク制約要素密度がそれぞれ p, q となるようにランダムに決定される．

p に関してPCOPとするために $m+1$ 個のノードのついてはすべてにノード間にアークが存在するようにし，さらに全体が一つのネットワークになるようにすべてのノードが接続されるようにするため，以下のように制限してある．

$$p \geq \frac{\frac{(m+1)m}{2} + n - m - 1}{\frac{n(n-1)}{2}} \tag{6.1}$$

q はn色問題となるようにするため，以下のように制限してある．

$$q \leq \frac{m-1}{m} \tag{6.2}$$

重みに関しては，ノードの要素・アーク制約要素・アーク制約の重み

が個別に定められ，それぞれに対し上限と下限が指定でき，指定された範囲内のランダムな重みが設定される．

6.3 RS解法システムの評価

6.3.1 比較する解法システムの実装

計算時間・解の値の比較評価のため，全解探索により最適解が得られる解構成型アルゴリズムとして木探索法 (PBB) の実装を行い，近似解がえられる状態空間型アルゴリズムとしてリスタート法 (GSAT) の実装を行った．PBB 及び GSAT の処理系はファイル読み込み・探索を含めて C を用いて記述した．

PBB に関して現在の実装で現実的な時間内で解が得られるのは，ノードに基づく帰着法を用いてノード数が 10 ・ノード要素数が 5 が限度である．そのため PBB との比較を行う評価は，ノード数を 10 ・ノード要素数を 5 とし，ノードの基づく帰着法を用いて行った．

GSAT との比較を行う評価については，ノード数を 20 ・ノード要素数を 10 とし，ノードの基づく帰着法を用いて行った．

問題の生成には n 色問題を用いた．解コスト・計算時間は問題の生成パラメータである n, m, p, q が同じ問題を 10 問生成し，それぞれの結果の平均値を用いた．

以下の実験結果では，単体法の反復適用による解に関しては RS(Repeat use of Simplex method) とし，PBB による解に関しては BB とし，GSAT による解に関しては GSAT として示す．

6.3.2 丸めの境界値による変化

「丸めの境界値」を決定するため，丸めの境界値を変化させたときの処理結果について比較を行った．

ノードの数を 10 とし，ノードの要素数を 5 とし，アーク密度・アーク制約要素密度を変化させたときの解コストと計算時間を求めた．ノードの値の重みを 1 とし，アークの重みを 100 とした．計算は PentiumPro200MHz の PC 上で行った．

ノード i について実数解と丸め先との距離 ND_i の最大値は，ノードの要素の数 m により変化する． ND_i が最も大きくなるのは，ノードの各要素の値に対応する変数 x_{ik} がすべて等しくなる場合，すなわち $x_{ik} = \frac{1}{m}$ となる場合である．この時の ND_i は以下のようになる．

$$\begin{aligned} ND_i &= \sqrt{\sum_{k\neq 0}(\frac{1}{m})^2 + (1-\frac{1}{m})^2} \\ &= \sqrt{\frac{m-1}{m^2} + 1 + \frac{1}{m^2} - \frac{2}{m}} \\ &= \sqrt{1-\frac{1}{m}} \end{aligned}$$

よって m が変化したとき ND_i の最大値がとり得る範囲は， $m = 2$ で $x_{i0} = x_{i1} = 0.5$ のときに最小となり， $ND_i = \sqrt{0.5} \simeq 0.707$ となる．よって，丸めの境界値をこの値より小さくすれば，丸めの操作による丸め先は最も近い位置であることが保証される． ND_i を変化させたときの丸めの境界値と丸めが行われる実数解の領域との関係は図 6.4 のようになる．

結果を表 6.1，表 6.2 に示し，丸めの境界値毎の解コスト・計算時間の平均値を，丸めの境界値が 0.0 の解コスト・計算時間の平均値で割った

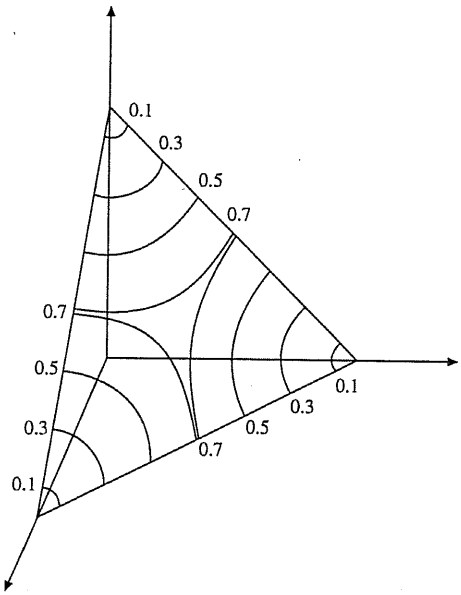

図 6.4: 丸めの境界値と実数解の領域との関係

値の，丸めの境界値による変化を図 6.5 に示す．

6.3.3 ノード数による変化

ノードの数 n を変化させたときの計算時間の測定を行った．

ノードの要素数を 5 とし，アーク密度を 0.3 とし，アーク制約要素密度を 0.8，各ノードの要素の数を 5 とし，ノードの数を変化させたときの計算時間を求めた．ノードの値の重みは 1 から 10 の範囲でランダムに生成し，アークの重みは 100 から 200 の範囲でランダムに生成した．計算は PentiumPro200MHz の PC 上で行った．

結果を表 6.3，図 6.6 に示す．それぞれのノードの数での ILP の制約数を表 6.3 の制約数とし，一回毎の単体法の計算時間の推移と反復適用法

アーク密度	アーク制約要素密度	丸めの境界値							
		0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0.511	0.120	1320.0	1320.0	1320.0	1320.0	1320.0	1320.0	1320.0	1330.0
0.511	0.200	870.0	870.0	870.0	870.0	870.0	870.0	880.0	920.0
0.511	0.320	470.0	470.0	470.0	470.0	470.0	490.0	480.0	580.0
0.511	0.400	390.0	390.0	390.0	390.0	390.0	390.0	370.0	430.0
0.511	0.520	180.0	180.0	180.0	180.0	180.0	180.0	180.0	400.0
0.511	0.600	170.0	170.0	170.0	170.0	170.0	170.0	200.0	320.0
0.511	0.720	110.0	110.0	110.0	110.0	110.0	110.0	110.0	180.0
0.511	0.800	110.0	110.0	110.0	110.0	110.0	110.0	110.0	150.0
0.600	0.120	1610.0	1610.0	1610.0	1610.0	1610.0	1610.0	1610.0	1640.0
0.600	0.200	1120.0	1120.0	1120.0	1120.0	1120.0	1120.0	1120.0	1150.0
0.600	0.320	710.0	710.0	710.0	710.0	710.0	710.0	720.0	740.0
0.600	0.400	540.0	540.0	540.0	540.0	540.0	540.0	540.0	580.0
0.600	0.520	250.0	250.0	250.0	250.0	270.0	300.0	290.0	440.0
0.600	0.600	170.0	170.0	170.0	170.0	170.0	170.0	210.0	380.0
0.600	0.720	120.0	120.0	120.0	120.0	120.0	140.0	130.0	230.0
0.600	0.800	110.0	110.0	110.0	110.0	110.0	110.0	110.0	110.0
0.689	0.120	1820.0	1820.0	1820.0	1820.0	1820.0	1820.0	1820.0	1830.0
0.689	0.200	1340.0	1340.0	1340.0	1340.0	1340.0	1340.0	1340.0	1380.0
0.689	0.320	920.0	920.0	920.0	920.0	920.0	920.0	920.0	1010.0
0.689	0.400	660.0	660.0	660.0	660.0	660.0	670.0	670.0	720.0
0.689	0.520	430.0	430.0	430.0	430.0	430.0	420.0	430.0	490.0
0.689	0.600	270.0	270.0	270.0	270.0	270.0	270.0	290.0	470.0
0.689	0.720	130.0	130.0	130.0	130.0	130.0	130.0	140.0	280.0
0.689	0.800	110.0	110.0	110.0	110.0	110.0	110.0	110.0	110.0
0.800	0.120	2360.0	2360.0	2360.0	2360.0	2360.0	2360.0	2360.0	2360.0
0.800	0.200	1840.0	1840.0	1840.0	1840.0	1840.0	1840.0	1840.0	1850.0
0.800	0.320	1120.0	1120.0	1120.0	1120.0	1120.0	1120.0	1130.0	1140.0
0.800	0.400	860.0	860.0	860.0	860.0	860.0	870.0	870.0	920.0
0.800	0.520	510.0	510.0	510.0	510.0	510.0	510.0	510.0	640.0
0.800	0.600	380.0	380.0	380.0	380.0	380.0	400.0	380.0	480.0
0.800	0.720	250.0	250.0	250.0	250.0	250.0	240.0	230.0	550.0
0.800	0.800	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0
0.911	0.120	2570.0	2570.0	2570.0	2570.0	2570.0	2570.0	2570.0	2570.0
0.911	0.200	1950.0	1950.0	1950.0	1950.0	1950.0	1960.0	1960.0	1950.0
0.911	0.320	1470.0	1470.0	1470.0	1470.0	1470.0	1470.0	1470.0	1500.0
0.911	0.400	1060.0	1060.0	1060.0	1060.0	1060.0	1060.0	1060.0	1080.0
0.911	0.520	630.0	630.0	630.0	630.0	630.0	630.0	630.0	690.0
0.911	0.600	480.0	480.0	480.0	480.0	480.0	480.0	500.0	760.0
0.911	0.720	340.0	340.0	340.0	340.0	340.0	330.0	330.0	560.0
0.911	0.800	270.0	270.0	270.0	270.0	270.0	270.0	270.0	270.0
1.000	0.120	2880.0	2880.0	2880.0	2880.0	2880.0	2880.0	2880.0	2890.0
1.000	0.200	2220.0	2220.0	2220.0	2220.0	2220.0	2220.0	2220.0	2250.0
1.000	0.320	1670.0	1670.0	1670.0	1670.0	1670.0	1670.0	1680.0	1720.0
1.000	0.400	1240.0	1240.0	1240.0	1240.0	1240.0	1240.0	1260.0	1380.0
1.000	0.520	800.0	800.0	800.0	800.0	800.0	800.0	800.0	910.0
1.000	0.600	650.0	650.0	650.0	650.0	650.0	660.0	630.0	740.0
1.000	0.720	520.0	520.0	520.0	520.0	520.0	520.0	510.0	760.0
1.000	0.800	510.0	510.0	510.0	510.0	510.0	510.0	510.0	510.0

表 6.1: 丸めの境界値の変化による解コストの変化

アーク密度	アーク制約要素密度	丸めの境界値							
		0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0.511	0.120	1.294	1.290	1.311	1.296	1.303	1.297	1.292	0.544
0.511	0.200	1.040	1.045	1.050	1.034	1.154	0.978	0.873	0.528
0.511	0.320	0.820	0.817	0.820	0.831	0.814	0.700	0.487	0.284
0.511	0.400	1.292	1.271	1.285	1.265	1.274	1.112	0.998	0.261
0.511	0.520	0.871	0.877	0.875	0.875	0.910	0.523	0.329	0.274
0.511	0.600	1.112	1.118	1.128	1.114	1.062	0.515	0.495	0.188
0.511	0.720	1.229	1.211	1.209	1.186	1.068	0.884	0.889	0.586
0.511	0.800	1.840	1.864	1.846	1.843	1.701	1.804	1.706	1.680
0.600	0.120	2.722	2.713	2.762	2.591	2.422	2.420	2.461	1.972
0.600	0.200	1.783	1.678	1.676	1.676	1.675	1.449	1.389	0.906
0.600	0.320	1.581	1.585	1.596	1.586	1.578	1.409	1.050	0.626
0.600	0.400	1.681	1.700	1.675	1.761	1.654	1.575	1.386	0.737
0.600	0.520	1.103	1.078	1.074	1.068	1.133	0.971	0.875	0.454
0.600	0.600	1.177	1.179	1.179	1.182	1.144	0.883	0.410	0.273
0.600	0.720	1.508	1.602	1.656	1.729	1.510	0.768	0.491	0.313
0.600	0.800	2.391	2.371	2.384	2.366	2.356	2.467	2.426	2.422
0.689	0.120	2.870	2.822	2.866	2.844	2.886	2.847	2.847	2.205
0.689	0.200	2.332	2.346	2.329	2.349	2.395	2.097	1.976	1.383
0.689	0.320	2.214	2.227	2.195	2.177	2.126	2.062	1.919	0.993
0.689	0.400	2.521	2.509	2.537	2.510	2.535	1.951	1.808	0.700
0.689	0.520	2.302	2.200	2.190	2.210	2.181	1.908	1.729	1.042
0.689	0.600	1.625	1.615	1.637	1.639	1.642	1.394	0.734	0.373
0.689	0.720	1.719	1.716	1.722	1.713	1.732	0.946	0.848	0.598
0.689	0.800	2.691	2.715	2.718	2.683	2.610	2.606	2.599	2.541
0.800	0.120	5.437	5.439	5.488	5.517	5.408	5.396	5.382	5.032
0.800	0.200	3.352	3.359	3.368	3.366	3.348	3.291	3.000	2.486
0.800	0.320	2.925	2.879	2.900	2.873	2.894	2.618	2.451	1.406
0.800	0.400	3.522	3.525	3.531	3.532	3.514	3.493	2.996	1.825
0.800	0.520	2.424	2.475	2.445	2.430	2.414	2.155	1.651	0.790
0.800	0.600	2.654	2.777	2.676	2.659	2.881	2.400	1.367	0.397
0.800	0.720	2.374	2.377	2.416	2.360	2.433	1.659	1.509	0.182
0.800	0.800	3.159	3.122	3.141	3.105	3.118	3.111	3.141	3.116
0.911	0.120	6.373	6.388	6.359	6.378	6.367	6.422	6.530	6.362
0.911	0.200	5.474	5.512	5.499	5.485	5.433	5.165	5.067	3.623
0.911	0.320	4.367	4.228	4.291	4.299	4.165	3.996	3.810	2.170
0.911	0.400	3.834	3.803	3.854	3.794	3.851	3.590	3.348	1.491
0.911	0.520	3.824	3.635	3.613	3.546	3.585	3.545	3.183	1.454
0.911	0.600	2.624	2.650	2.647	2.428	2.412	1.972	1.527	0.704
0.911	0.720	2.832	2.741	2.775	2.748	2.741	2.415	2.158	0.768
0.911	0.800	4.113	3.997	4.250	4.031	4.392	3.885	4.066	4.202
1.000	0.120	10.469	10.287	10.144	10.009	10.097	10.333	9.977	9.082
1.000	0.200	6.172	6.119	6.046	6.002	5.946	5.998	5.898	4.470
1.000	0.320	5.851	5.849	5.850	5.806	5.827	5.688	5.283	3.165
1.000	0.400	4.996	4.969	4.931	5.007	4.953	4.463	4.293	2.303
1.000	0.520	5.090	5.065	5.057	5.038	5.070	4.889	4.811	1.532
1.000	0.600	3.598	3.635	3.624	3.597	3.537	3.192	2.497	0.612
1.000	0.720	3.357	3.373	3.362	3.368	3.376	2.738	1.926	0.518
1.000	0.800	4.397	4.396	4.434	4.302	4.398	4.372	4.459	4.389

表 6.2: 丸めの境界値の変化による計算時間の変化 (時間の単位は [sec])

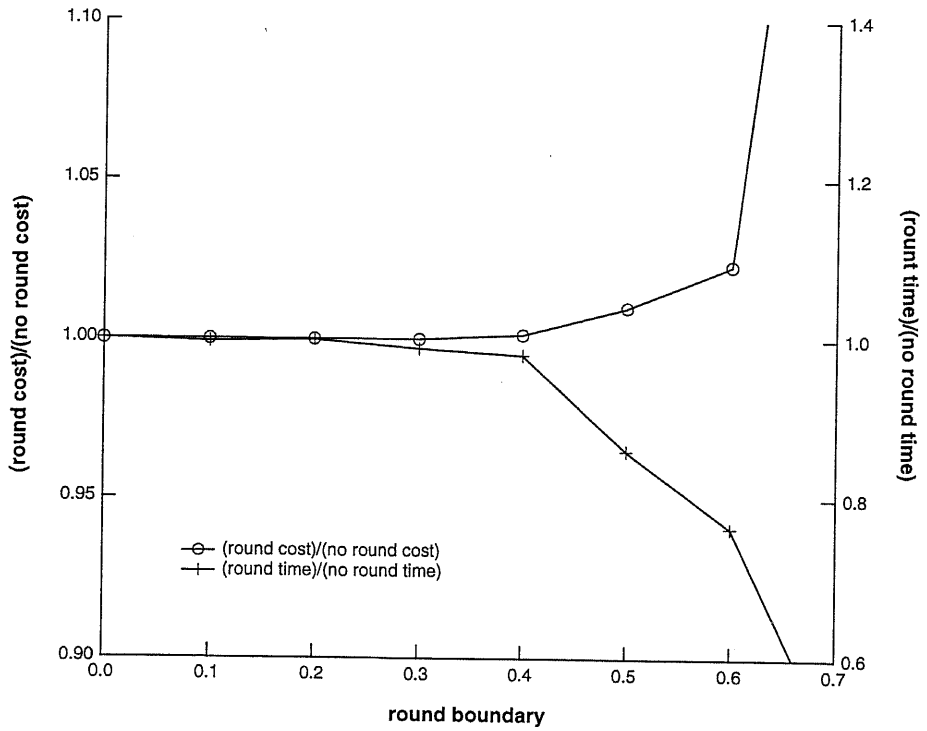

図 6.5: 丸めの境界値の変化による解コストと計算時間の変化

の計算を表 6.3 の 1 回目から 61 回目と合計に示した．図 6.6 では反復適用法の計算時間を RS とし，比較のために解構成型アルゴリズムによる計算時間を BB とし，ノードの数による変化を示した．

全体の傾向としてはノードの数が増えると急に計算時間が延る．しかしながら一回毎の単体法の計算時間を見ていくと，合計の計算時間の延びと比較してそれほど急には延ていないことが分かる．

ノード数	20	30	40	50	60	70	80
制約数	590	1340	2380	3730	5370	7320	9560
1回目	0.15	0.31	0.86	2.42	3.52	5.32	9.66
11回目			1.72	3.49	8.53	16.48	19.32
21回目				3.28	8.40	13.49	23.87
31回目					6.73	12.97	22.31
41回目					5.69	12.32	26.19
51回目						7.40	24.78
61回目							9.73
合計	0.24	1.65	21.44	77.49	281.44	688.74	1493.67

表 6.3: ノード数の変化による計算時間の変化 (時間の単位は [sec])

6.3.4 解構成型アルゴリズムとの比較

解コスト・計算時間について解構成型アルゴリズムである PBB との比較を行った．ノードの数を 10 とし，ノードの要素数を 5 とし，アーク密度・アーク制約要素密度を変化させたときの解コストの比と計算時間を求めた．ノードの値の重みを 1 とし，アークの重みを 100 とした．計算は PentiumPro200MHz の PC 上で行った．

結果を表 6.4，図 6.7，図 6.8 に示す．

解コストの比 $\frac{RScost}{BBcost}$ の全体の平均は 1.16 となり，560 問中 192 問 (34.3%) で最適解である木探索法と同じ解が得られた．解コスト比はアーク制約要素密度が高いもしくは低い場合に低くなり，ほぼ 0.6 付近で悪い値をとっている．計算時間に関しては一部 BB より遅くなっているのが

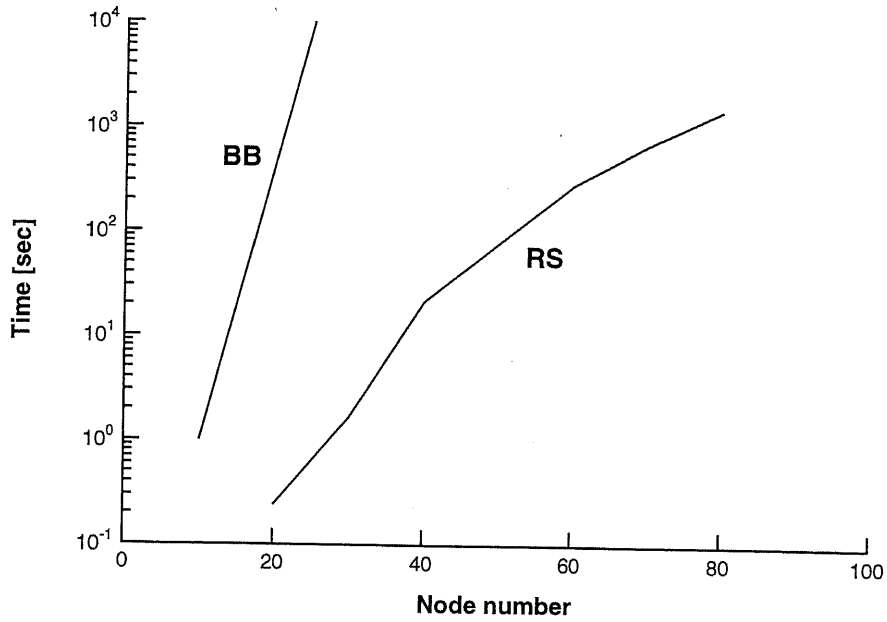

図 6.6: ノード数の変化による計算時間の変化

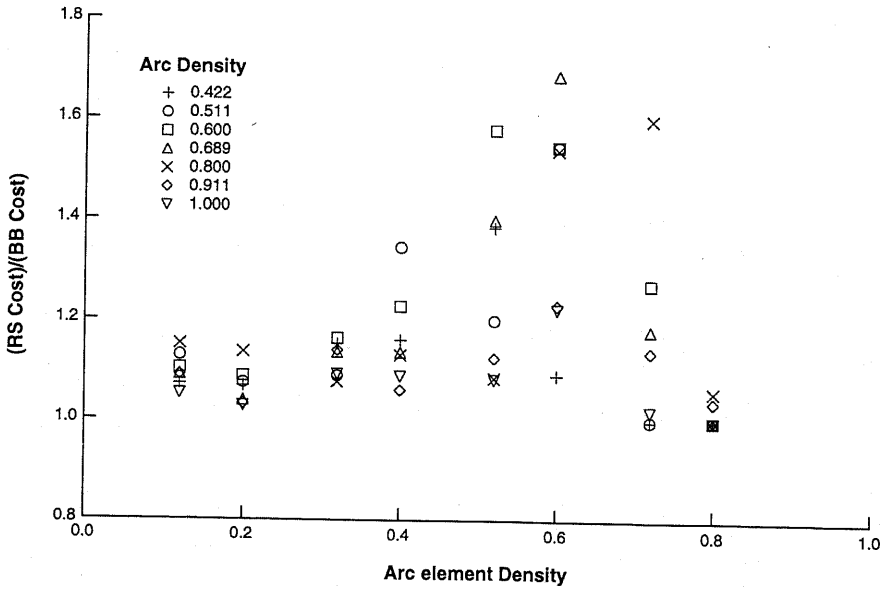

図 6.7: 解構成型アルゴリズムとの解コストの比の変化

アーク密度	アーク制約要素密度	$RScost$	$BBcost$	$\frac{RScost}{BBcost}$	$RStime$	$BBtime$	$\frac{RStime}{BBtime}$
0.511	0.120	1320.0	1170.0	1.1282	1.297	7.956	0.1630
0.511	0.200	870.0	810.0	1.0741	0.978	4.662	0.2098
0.511	0.320	490.0	450.0	1.0889	0.700	3.659	0.1913
0.511	0.400	390.0	290.0	1.3448	1.112	1.821	0.6107
0.511	0.520	180.0	150.0	1.2000	0.523	1.325	0.3947
0.511	0.600	170.0	110.0	1.5455	0.515	1.210	0.4256
0.511	0.720	110.0	110.0	1.0000	0.884	3.326	0.2658
0.511	0.800	110.0	110.0	1.0000	1.804	18.767	0.0961
0.600	0.120	1610.0	1460.0	1.1027	2.420	17.753	0.1363
0.600	0.200	1120.0	1030.0	1.0874	1.449	10.173	0.1424
0.600	0.320	710.0	610.0	1.1639	1.409	3.313	0.4253
0.600	0.400	540.0	440.0	1.2273	1.575	3.495	0.4506
0.600	0.520	300.0	190.0	1.5789	0.971	1.273	0.7628
0.600	0.600	170.0	110.0	1.5455	0.883	1.160	0.7612
0.600	0.720	140.0	110.0	1.2727	0.768	2.140	0.3589
0.600	0.800	110.0	110.0	1.0000	2.467	8.141	0.3030
0.689	0.120	1820.0	1670.0	1.0898	2.847	14.661	0.1942
0.689	0.200	1340.0	1290.0	1.0388	2.097	13.531	0.1550
0.689	0.320	920.0	810.0	1.1358	2.062	4.977	0.4143
0.689	0.400	670.0	590.0	1.1356	1.951	2.683	0.7272
0.689	0.520	420.0	300.0	1.4000	1.908	1.988	0.9598
0.689	0.600	270.0	160.0	1.6875	1.394	0.993	1.4038
0.689	0.720	130.0	110.0	1.1818	0.946	1.149	0.8233
0.689	0.800	110.0	110.0	1.0000	2.606	5.443	0.4788
0.800	0.120	2360.0	2050.0	1.1512	5.396	16.794	0.3213
0.800	0.200	1840.0	1620.0	1.1358	3.291	13.425	0.2451
0.800	0.320	1120.0	1040.0	1.0769	2.618	7.698	0.3401
0.800	0.400	870.0	770.0	1.1299	3.493	5.184	0.6738
0.800	0.520	510.0	470.0	1.0851	2.155	2.849	0.7564
0.800	0.600	400.0	260.0	1.5385	2.400	1.852	1.2959
0.800	0.720	240.0	150.0	1.6000	1.659	2.010	0.8254
0.800	0.800	180.0	170.0	1.0588	3.111	6.270	0.4962
0.911	0.120	2570.0	2360.0	1.0890	6.422	20.125	0.3191
0.911	0.200	1960.0	1900.0	1.0316	5.165	16.559	0.3119
0.911	0.320	1470.0	1290.0	1.1395	3.996	8.694	0.4596
0.911	0.400	1060.0	1000.0	1.0600	3.590	6.802	0.5278
0.911	0.520	630.0	560.0	1.1250	3.545	4.093	0.8661
0.911	0.600	480.0	390.0	1.2308	1.972	2.917	0.6760
0.911	0.720	330.0	290.0	1.1379	2.415	3.453	0.6994
0.911	0.800	270.0	260.0	1.0385	3.885	9.818	0.3957
1.000	0.120	2880.0	2740.0	1.0511	10.333	27.401	0.3771
1.000	0.200	2220.0	2160.0	1.0278	5.998	20.233	0.2964
1.000	0.320	1670.0	1530.0	1.0915	5.688	12.661	0.4493
1.000	0.400	1240.0	1140.0	1.0877	4.463	7.821	0.5706
1.000	0.520	800.0	740.0	1.0811	4.889	5.334	0.9166
1.000	0.600	660.0	540.0	1.2222	3.192	3.955	0.8071
1.000	0.720	520.0	510.0	1.0196	2.738	12.346	0.2218
1.000	0.800	510.0	510.0	1.0000	4.372	78.387	0.0558

表 6.4: 解構成型アルゴリズムとの比較 (時間の単位は [sec])

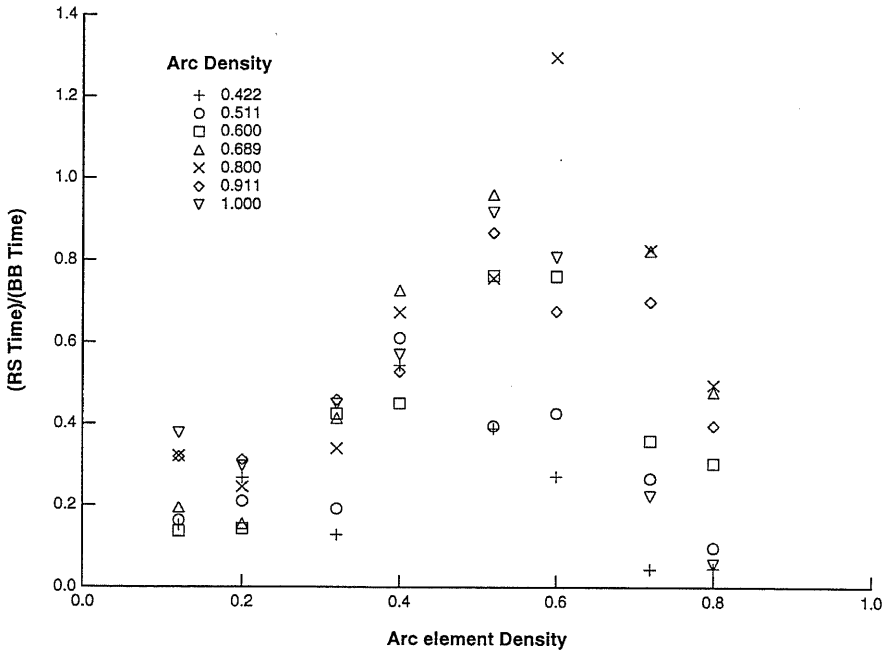

図 6.8: 解構成型アルゴリズムとの計算時間の比の変化

あるが大部分はBBよりも速くなっており，特にアーク制約要素密度が高いもしくは低い場合に速くなる．

6.3.5 状態空間型アルゴリズムとの比較

解コスト・計算時間について状態空間型アルゴリズムであるGSATとの比較を行った．ノードの数を20とし，ノードの要素数を10とし，アーク密度・アーク制約要素密度を変化させたときの解コストの比と計算時間を求めた．ノードの値の重みは 1 から 10 の範囲でランダムに生成し，アークの重みは100から200の範囲でランダムに生成した．計算はPentiumII333MHzのPC上で行った．

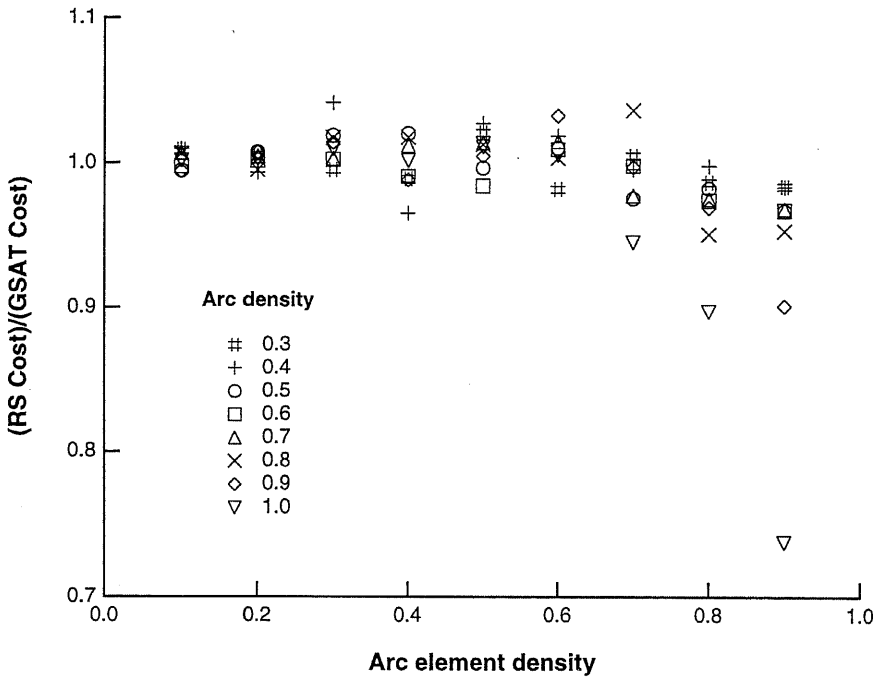

図 6.9: 状態空間型アルゴリズムとの解コストの比の変化

結果を表 6.5，表 6.6，図 6.9，図 6.10 に示す．

解コストの比 $\frac{RScost}{GSATcost}$ の全体の平均は 0.9919 となった．解コスト比はアーク制約要素密度が高い場合に低くなり，低くなるにつれ悪くなる．アーク密度が変ってもそれほど変化しないが，アーク密度が高くアーク制約要素密度が高い問題ではやや低くなっている．

計算時間はアーク制約要素密度が高い問題では 1 桁から 2 桁ほど速くなり，低くなるにつれ遅くなり同じ程度か 10 倍程度まで遅くなる．またアーク密度が低い問題では速く高い問題では遅くなり，アーク制約要素密度が高くアーク密度が低い問題では 2 桁ほど速くなるのに対し，アーク制約要素密度が低くアーク密度が高い問題では 1 桁ほど遅くなる．

アーク密度	アーク制約要素密度	$RScost$	$GSATcost$	$\frac{RScost}{GSATcost}$	$RStime$	$GSATtime$	$\frac{RStime}{GSATtime}$
0.337	0.100	2649.0	2625.6	1.0089	110.563	138.902	0.7960
0.337	0.200	2014.6	2005.9	1.0043	156.977	234.452	0.6695
0.337	0.300	1495.2	1501.6	0.9957	93.260	325.238	0.2867
0.337	0.400	1196.2	1208.9	0.9895	50.973	432.872	0.1178
0.337	0.500	974.7	954.4	1.0213	35.569	528.519	0.0673
0.337	0.600	715.2	728.0	0.9824	19.356	605.718	0.0320
0.337	0.700	585.7	582.6	1.0053	9.069	660.529	0.0137
0.337	0.800	493.2	499.7	0.9870	9.159	706.785	0.0130
0.337	0.900	421.1	428.0	0.9839	4.776	717.226	0.0067
0.400	0.100	3062.3	3033.3	1.0096	177.575	168.969	1.0509
0.400	0.200	2335.2	2350.1	0.9937	245.795	287.507	0.8549
0.400	0.300	1784.0	1712.6	1.0417	184.335	421.515	0.4373
0.400	0.400	1336.4	1383.7	0.9658	117.850	539.507	0.2184
0.400	0.500	1109.3	1079.5	1.0276	70.809	641.417	0.1104
0.400	0.600	852.3	836.4	1.0190	28.151	725.598	0.0388
0.400	0.700	631.7	634.1	0.9962	20.329	811.170	0.0251
0.400	0.800	541.2	542.2	0.9982	13.590	852.657	0.0159
0.400	0.900	439.4	446.7	0.9837	4.666	903.775	0.0052
0.500	0.100	3545.5	3563.9	0.9948	275.515	219.337	1.2561
0.500	0.200	2916.2	2894.5	1.0075	427.005	373.654	1.1428
0.500	0.300	2285.1	2242.3	1.0191	385.150	577.350	0.6671
0.500	0.400	1761.4	1726.1	1.0205	271.560	723.015	0.3756
0.500	0.500	1204.9	1208.7	0.9969	156.114	838.345	0.1862
0.500	0.600	935.0	924.8	1.0110	82.321	926.059	0.0889
0.500	0.700	708.9	726.3	0.9760	41.351	1001.682	0.0413
0.500	0.800	557.5	567.3	0.9827	29.180	1083.471	0.0269
0.500	0.900	437.4	452.2	0.9673	11.407	1116.603	0.0102
0.600	0.100	4487.3	4497.6	0.9977	510.163	265.524	1.9213
0.600	0.200	3473.8	3468.6	1.0015	688.288	504.429	1.3645
0.600	0.300	2698.6	2691.7	1.0026	599.889	728.755	0.8232
0.600	0.400	1992.9	2011.0	0.9910	431.642	903.046	0.4780
0.600	0.500	1500.4	1524.0	0.9845	297.593	1018.633	0.2921
0.600	0.600	1093.1	1082.5	1.0098	215.563	1123.329	0.1919
0.600	0.700	830.2	831.4	0.9986	125.917	1192.982	0.1055
0.600	0.800	572.7	587.6	0.9746	67.246	1260.885	0.0533
0.600	0.900	467.8	483.4	0.9677	20.563	1325.845	0.0155

表 6.5: 状態空間型アルゴリズムとの比較1(時間の単位は [sec])

アーク密度	アーク制約要素密度	$RScost$	$GSATcost$	$\frac{RScost}{GSATcost}$	$RStime$	$GSATtime$	$\frac{RStime}{GSATtime}$
0.700	0.100	5255.5	5220.7	1.0067	1014.705	317.564	3.1953
0.700	0.200	4124.1	4104.8	1.0047	995.075	617.412	1.6117
0.700	0.300	3246.0	3235.2	1.0033	910.828	868.924	1.0482
0.700	0.400	2449.7	2419.8	1.0124	694.373	1057.590	0.6566
0.700	0.500	1847.1	1821.1	1.0143	472.660	1210.177	0.3906
0.700	0.600	1343.2	1322.8	1.0154	304.417	1320.364	0.2306
0.700	0.700	923.8	944.6	0.9780	133.073	1401.550	0.0949
0.700	0.800	658.4	674.9	0.9756	62.417	1461.211	0.0427
0.700	0.900	479.1	494.6	0.9687	42.467	1546.659	0.0275
0.800	0.100	5951.4	5914.2	1.0063	1724.931	363.676	4.7430
0.800	0.200	4773.4	4795.5	0.9954	1318.924	737.567	1.7882
0.800	0.300	3712.0	3647.9	1.0176	1211.879	1011.370	1.1983
0.800	0.400	2882.2	2831.3	1.0180	985.805	1213.096	0.8126
0.800	0.500	2188.0	2159.7	1.0131	698.157	1387.029	0.5033
0.800	0.600	1549.3	1543.4	1.0038	503.626	1519.154	0.3315
0.800	0.700	1140.3	1100.2	1.0364	358.871	1607.164	0.2233
0.800	0.800	722.2	759.2	0.9513	137.771	1668.132	0.0826
0.800	0.900	531.3	557.2	0.9535	78.964	1693.285	0.0466
0.900	0.100	6860.3	6899.8	0.9943	3566.143	403.098	8.8468
0.900	0.200	5424.1	5383.9	1.0075	1859.883	851.709	2.1837
0.900	0.300	4438.1	4374.8	1.0145	1737.343	1143.344	1.5195
0.900	0.400	3336.9	3375.1	0.9887	1446.461	1390.860	1.0400
0.900	0.500	2512.9	2499.9	1.0052	972.331	1573.795	0.6178
0.900	0.600	1850.7	1791.7	1.0329	986.278	1731.373	0.5697
0.900	0.700	1268.3	1271.3	0.9976	423.005	1820.746	0.2323
0.900	0.800	867.6	894.8	0.9696	282.532	1862.147	0.1517
0.900	0.900	619.2	686.9	0.9014	112.186	1940.409	0.0578
1.000	0.100	7704.4	7691.1	1.0017	4871.409	480.327	10.1419
1.000	0.200	6137.0	6119.6	1.0028	2295.396	963.815	2.3816
1.000	0.300	4852.1	4803.5	1.0101	2318.505	1281.303	1.8095
1.000	0.400	3795.5	3787.0	1.0022	1831.774	1555.369	1.1777
1.000	0.500	2880.8	2847.7	1.0116	1385.181	1762.624	0.7859
1.000	0.600	2081.2	2069.2	1.0058	1210.566	1938.006	0.6246
1.000	0.700	1420.1	1501.5	0.9458	629.744	2016.889	0.3122
1.000	0.800	996.9	1110.9	0.8974	337.544	1920.086	0.1758
1.000	0.900	759.8	1029.8	0.7378	166.613	882.539	0.1888

表 6.6: 状態空間型アルゴリズムとの比較2(時間の単位は [sec])

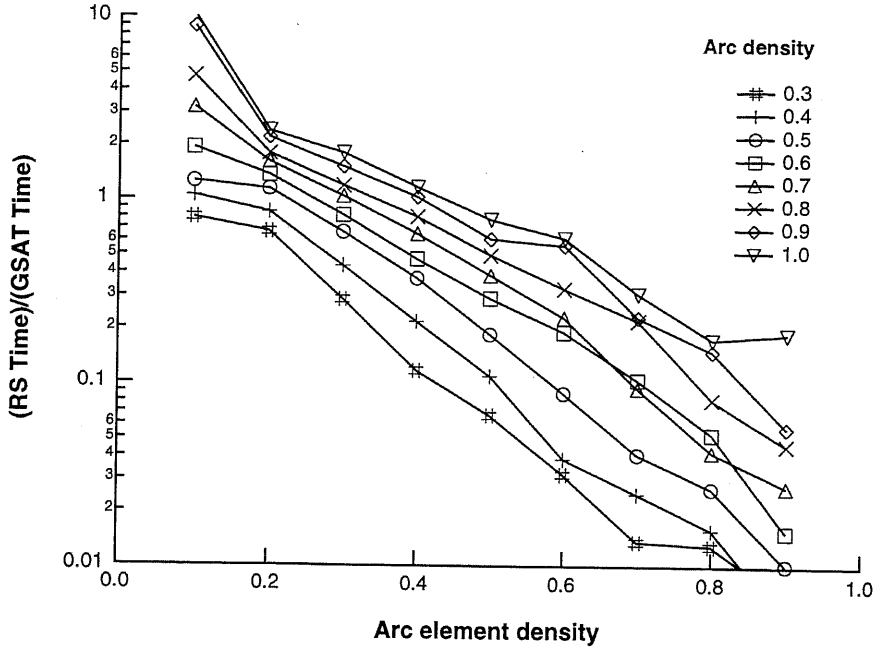

図 6.10: 状態空間型アルゴリズムとの計算時間の比の変化

6.4 考察

6.4.1 丸めの境界値による変化

結果より丸めの境界値としては 0.5 を用いれば良いことがわかる.

丸めの境界値による解の質は 0.7 以外ではほとんど変らず, 計算時間も 0.7 以外では劇的な変化はない. つまり, 丸めの境界値が 0.5 近辺では, 丸めの境界値による解の質・計算時間の変化は小さいと言える.

解の質・計算時間の結果だけではなく, パラメータの値の変化による解の質・計算時間に対しての影響の程度が低く多少の誤差の影響が少ないという点からも, 丸めの境界値として 0.5 を用いることは妥当であると言える.

6.4.2 ノード数による変化

ノード数が増えると計算時間が急に延びるのに対し，一回毎の単体法の計算時間はそれほど延びていない．これは計算時間の合計が延るのは，最終解を得るまでに必要とされる単体法の計算回数が増えることによるものと考えられる．

表 6.3 の実験はアーク密度を固定して行っている．この場合一つのノードに関与するアークの数は，ノードの数が増えるにしたがって増加する．そのためノード間の結合が強くなり，全体としては制約がきつくなると考えられる．

単体法の計算時間は単体法の制約の数もしくは変数の数により変化するが，計算時間のオーダはほとんどの問題において制約の数や変数の数の多項式オーダーであると考えられる．一方単体法の繰り返し回数は，ノードの数を n としたときノードの決定までに必要な繰り返し回数の最大値は一つのノードが決定されるまで最大でもノードの数分だけ計算するだけで決定されるので，$\frac{n(n-1)}{2}$ となる．そのため単体法の繰り返し回数のオーダーは多項式オーダとなる．以上の関係により本手法の計算時間はほとんどの問題において多項式オーダーであることが保証される．

図 6.6 ではノード数が増えるに従い，計算時間の延びが緩くなっている．計算時間が指数オーダの場合はノード数と計算時間の関係はグラフ上では直線として表される．このことからも解が指数オーダより速く得られていることが示される．

6.4.3 解構成型アルゴリズムとの比較

通常のCSPでは，アーク制約要素密度が高い場合は問題の制約が緩くなり解が多く存在し，解の探索は容易になる．またアーク制約要素密度が低い場合は問題の制約がきつくなり解はほとんど存在しなくなるが，制約伝播が発生することにより解の探索は容易になる．これに対し，アーク制約要素密度か中間値をとるときは，解の存在数は少なくなるにも関わらず制約伝播が発生しにくくなり，解の探索が困難になる．このような現象のことはCSPの相転移と呼ばれる．

PCOPでは，制約の不成立が認められているため，解が一意に決定される単純な制約伝播は発生しない．PCOPの幾つかのノードが決定すると決定したノード間のアークについてはアークの成立不成立が決定する．PCOPの解コストは決定したノードとアークのコストの和より大きくなる．そのため，解コストを小さくするには，他のノードの値のとり得る値も限定され，あたかも制約伝播が発生したがごとく他のノードの値も決定される．

このためPCOPではアーク制約要素密度が低い場合でも解の探索は比較的容易となる．アーク制約要素密度が高い場合は問題の制約が緩くなるため，解候補が数多く存在し，解の探索は容易になる．これに対し，アーク制約要素密度が中間値をとるときは，CSP同様に相転移が発生し解の探索が困難になる．

PBBとの比較で解コストがアーク制約要素密度が高いもしくは低い場合に低くなり，ほぼ0.6付近で悪い値をとっているのはPCOPの相転移が発生しているためと考えられる．

計算時間に関しては一部BBより遅くなっているのは，この評価はノード数を10としており，PBBでも解けるように問題規模を小さくしているためと思われる．ノード数を変化による計算時間の変化はPBB法では指数オーダとなっているため，実際の問題のように評価問題より大きいな問題に関してはPBBとには大きな開きが生じるものと考えられる．

6.4.4 状態空間型アルゴリズムとの比較

解コスト比がアーク制約要素密度が低い問題で悪くなるのは，単体法の実数解が丸め先から離れているためである．

ここで実数解の質を示す値として整数度を用いる．ノード数を n とし，ノードの要素数を m とし，i 番目のノード k 番目の要素の実数解を x_{ik} としたとき，整数度 IR は以下のように定義される．

$$IR = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{m} x_{ik}^2$$

整数度は，すべての解が整数解である場合に整数度は1となり，すべての解が $1/m$ である場合に整数度は $1/m$ となる．整数度が大きければ大きいほど実数解が整数解に近いことを示している．

アーク密度とアーク制約要素密度による整数度の変化を表6.7に示す．

解の整数度のアーク制約要素密度による変化は，アーク制約要素密度が低い場合は低くなり，高い場合は高くなる．またアーク密度による変化は，それほど大きくはないが，アーク密度が低い場合は高くなり，アーク密度が高い場合は低くなる．

整数度は解のばらつきを示しており，整数度が低い解は大部分の実数解が同じ値をとっていることを示している．特に表6.7で示した実数

	アーク制約要素密度								
アーク密度	0.100	0.200	0.300	0.400	0.500	0.600	0.700	0.800	0.900
0.337	0.236	0.265	0.340	0.378	0.420	0.484	0.579	0.686	0.772
0.400	0.171	0.207	0.226	0.260	0.324	0.395	0.514	0.649	0.796
0.500	0.139	0.155	0.170	0.204	0.260	0.329	0.413	0.568	0.705
0.600	0.120	0.141	0.152	0.186	0.236	0.288	0.372	0.492	0.649
0.700	0.109	0.130	0.140	0.168	0.215	0.265	0.343	0.468	0.606
0.800	0.103	0.124	0.135	0.163	0.204	0.253	0.316	0.442	0.582
0.900	0.101	0.122	0.130	0.153	0.193	0.240	0.310	0.409	0.549
1.000	0.101	0.119	0.125	0.149	0.187	0.236	0.298	0.386	0.535

表 6.7: アーク密度とアーク制約要素密度による解の整数度の変化

解はノードの要素数 m が 10 であるので，整数度の最低値は 0.1 となる．表 6.7 ではアーク制約要素密度が低くアーク密度が高い場合，整数度が 0.1 近い値をとっており，実数解のばらつきがほとんど無いことを意味している．

このような場合ノードの丸めにおいて，丸めをノードの選択や丸め先の選択はわずかの差で決定される．そのため丸めに誤差が生じやすくなり，誤った丸めが選択されやすくなる．誤った丸めに対しては，山登り法による解の改善によりある程度は改善されるが，完全ではない．そのためランダムな探索である GSAT と比較した場合解コストが悪くなりやすい．

整数度が高い解は，実数解がばらついており大きな値が解に含まれていることを示す．このことは丸めが比較的決定的に行われ，かつ丸めに誤差が生じにくく，丸めに対する実数解の変動も少ないことを意味する．そのため丸めの結果も最適解の近傍になり，ランダムな探索である GSAT

	k									
反復回数	1	2	3	4	5	6	7	8	9	10
1	0.053	0.000	0.173	0.032	0.216	0.167	0.189	0.136	0.022	0.012
2	0.006	0.019	0.150	0.072	0.225	0.117	0.194	0.153	0.063	0.000
3	0.021	0.020	0.163	0.050	0.232	0.110	0.181	0.152	0.070	0.000
4	0.004	0.009	0.193	0.102	0.173	0.106	0.189	0.189	0.036	0.000
5	0.046	0.040	0.144	0.007	0.202	0.112	0.217	0.173	0.059	0.000
6	0.037	0.052	0.126	0.048	0.240	0.101	0.179	0.142	0.073	0.000
7	0.000	0.066	0.093	0.229	0.066	0.110	0.110	0.089	0.236	0.000
8	0.000	0.000	0.124	0.225	0.040	0.153	0.175	0.156	0.127	0.000
9	0.000	0.000	0.178	0.229	0.077	0.136	0.154	0.109	0.112	0.004
10	0.000	0.000	0.192	0.202	0.068	0.176	0.136	0.093	0.133	0.000
11	0.000	0.000	0.238	0.299	0.000	0.000	0.151	0.151	0.160	0.000
12	0.000	0.000	0.255	0.288	0.000	0.000	0.164	0.179	0.113	0.000
13	0.000	0.000	0.294	0.235	0.000	0.000	0.176	0.176	0.118	0.000
14	0.000	0.000	0.214	0.214	0.000	0.000	0.286	0.143	0.143	0.000
15	0.000	0.000	0.000	0.333	0.000	0.000	0.333	0.000	0.333	0.000
16	0.000	0.000	0.000	0.000	0.000	0.000	0.500	0.000	0.500	0.000
17	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000

表 6.8: RS 法による，あるノードの実数解の変化

よりは準最適解が得られやすく，良い解が得られる．

整数度がやや低い解は，実数解の一部がばらついていることを示す．このような解は丸めを行った場合，解の様子が大きく変化するため丸めによる影響が大きく，最適解が実数解から離れている場合が往々にして存在する．単体法の反復適用では丸めを行う毎に実数最適解を計算することで常に最適解の近傍の実数解が得られるため，単純な丸めと比較し局所最適に陥りにくくなっている．

整数度がやや低い解で実数解が丸めにより移っていく様子を示す．アーク密度が 0.400 ，アーク制約要素密度が 0.300 ，最初の単体法の解の整数度が 0.208 の PCOP についてあるノードについて反復適用毎の実数解の変化を表 6.8，図 6.11 に示す．

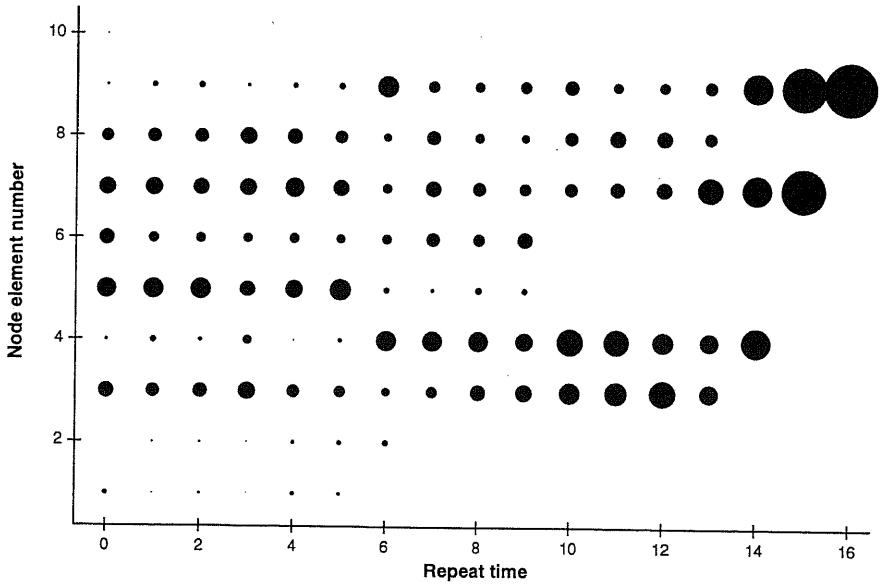

図 6.11: RS 法による，あるノードの実数解の変化

表 6.8，図 6.11 より反復適用毎に値が変化し，徐々に整数解近辺に移ってきていることがわかる．これにより局所最適を避けつつ準最適解が得られている．

GSAT との比較でアーク制約要素密度が低い問題で計算時間がかかるのは，基本的には単体法の一回当りの計算時間が延びているためである．最初の単体法の計算時間を表 6.9 に示す．

アーク密度による変化は，アーク制約要素密度によらず，アーク密度が高くなるにつれ長くかかるようになる．ILP の制約式の数はアークの数に比例する．そのためアーク密度が高くなると ILP の制約式が増え，その結果単体法の計算時間が延び，アーク密度が低い場合は逆になる．よってアーク密度による単体法の計算時間の変化は，制約式の数によるものと考えられる．

	アーク制約要素密度								
アーク密度	0.100	0.200	0.300	0.400	0.500	0.600	0.700	0.800	0.900
0.337	9.493	15.629	11.067	5.318	2.713	1.334	0.518	0.270	0.207
0.400	16.521	28.571	20.296	10.983	4.998	2.170	0.902	0.395	0.250
0.500	33.679	65.978	49.248	24.559	9.614	3.805	1.519	0.625	0.351
0.600	53.989	121.330	83.588	39.887	14.511	6.922	2.736	0.779	0.475
0.700	74.340	173.926	137.482	63.886	26.414	10.356	3.494	1.248	0.566
0.800	99.202	232.970	175.633	84.958	32.242	14.131	4.757	1.793	0.714
0.900	110.857	310.252	251.393	116.809	48.394	20.324	7.046	2.402	0.908
1.000	134.095	385.009	305.864	146.080	61.030	25.472	9.062	3.007	1.048

表 6.9: アーク密度とアーク制約要素密度による単体法の計算時間の変化

アーク制約要素密度による変化は，アーク制約要素密度が低い方が計算時間が長くなる．また最も計算時間が長くなるのは，アーク制約要素密度が最小の場合ではなく，それより大きな値である 0.2 のときである．

一般に単体法の計算時間は制約式の数に依存すると言われているが，アーク制約要素密度を変化させても ILP の制約式の数・変数の数は変化しない．この場合単体法の計算時間は問題の規模ではなく問題の複雑さの影響を受けていると考えられる．アーク制約要素密度が 0.2 近辺で相転移が発生し問題が複雑になり，単体法の計算時間が延びていると考えられる．

先に述べたようにアーク密度が高くアーク制約要素密度が低い場合は単体法の解の整数度が低くなる．そのため一度に多くの値を丸めることができなくなり，丸めの回数が増え単体法の計算回数が多くなる．逆にアーク密度が低くアーク制約要素密度が高い場合は単体法の解の整数度が高くなり，一度に多くの値を丸められるため単体法の計算回数が減る．図 6.12 に整数度と状態空間型アルゴリズムとの計算時間の比の関係を示す．

これにより GSAT と比較して，アーク制約要素密度が低くアーク密度

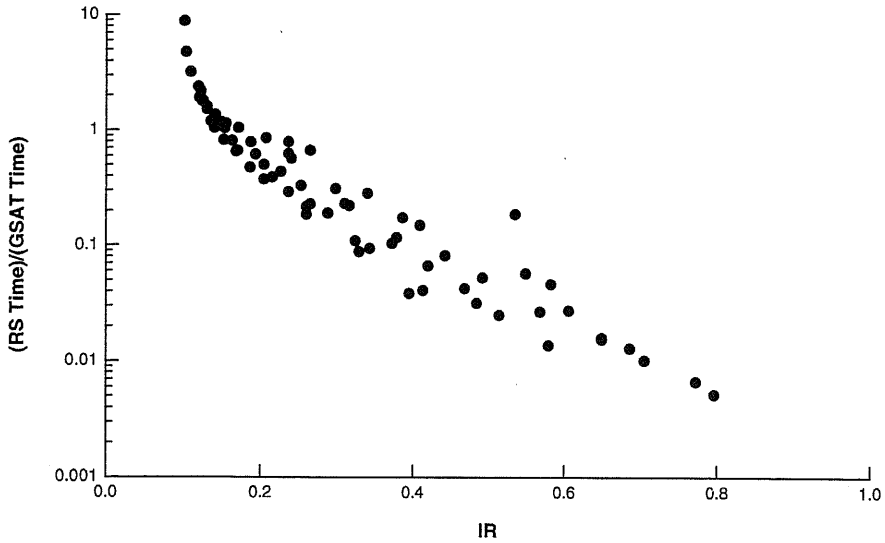

図 6.12: 整数度による状態空間型アルゴリズムとの計算時間の比の変化

が高い問題では計算時間が長くかかるようになり，逆にアーク制約要素密度が高くアーク密度が低い問題では計算時間が短くなる．また図 6.12 のように計算時間は解の整数度との相関が強く，整数度が高い問題では計算時間が短くなる．

6.5 RS 法の限界

PCOP は CSP の拡張と考えられるため，原理的には本稿の RS 法を用いることにより CSP を解くことも可能である．しかしながら 0–1 整数計画問題への帰着では成立不可能な制約を認めるように拡張を行い，RS 法では局所制約伝播による整合化を利用していないため，通常の CSP の解法に比べると解コスト・計算時間共に性能は悪いと考えられ適さない．同様に COP についてもあまり適さないと考えられる．

状態空間型アルゴリズムとの比較の結果から，制約のきつい問題にたいしても本手法は適さないと考えられる．PCOPは元々は制約のきついく解が得られない問題である．しかしながらPCOPに分類される問題すべてが制約のきつい問題であるとは限らない．

スケジューリング問題などで，最低限の要件だけを制約とした場合CSPとなるが，それ以外の要件もいれるとPCSPになる場合が存在する．例としては，当番割り当てで当番不可能な日を満たすだけならCSPとなるが，連続して当番に当たらないようにするとPCSPとなるような場合があげられる．このような場合，連続しないというのは多くの組合わせの中の一つでしかないため，制約としては非常に緩い制約である．そのため連続しないという制約をも含めたCSPはPCSPとなるにも関わらず，全体の制約は緩く，解法として本手法が適当となる．

逆に全体的に制約がきついためPCOPとなる問題に関しては本手法は適さない．実験結果からは最初の単体法の解の整数度がアークの平均要素数を m としたとき $2/m$ より小さい問題に関しては，GSATと比べ解コストこそそれほど大きく変らないものの，計算時間が一桁ほど遅くなる．そのためGSATなど別の手法を用いた方が，少なくとも計算時間の点からは有利であると考えられる．特に $1/m$ に近い問題に関しては，本手法の帰着法では最適解が単体法の解に反映されないため，ILPの解法だけではなく帰着法そのものを変更する必要がある．

大規模な問題で問題となるのが計算時間とメモリ容量である．6.3.3節よりノード数が80ノードになると20分以上かかっており，この時に必要とされるメモリサイズは数十MBである．この時必要とされたメモリの大部分はバッファに用いられているものであり，計算時間を多少犠牲

にすることによりより少なくすることは可能である．また近年メモリの容量は飛躍的に増加しているため，メモリによる制限はほとんどないものと考えられる．計算時間は計算機の性能に依存するが，実験に用いた計算機の性能は本稿執筆時に一般に市販されている計算機の性能よりやや悪い程度であり，その性能比はせいぜい数倍程度であると考えられる．またスーパーコンピュータなどを用いない限り多少高性能の計算機を用いても性能比は十数倍程度と考えられる．

実験で用いた条件はアーク密度が 0.3 でアーク制約要素密度が 0.8 であり，一つのノードから出ているアークの数は平均すると 24 本である．これは制約のきつさで考えると，実際の応用問題程度であるかややきつい問題であると考えられる．単体法の解法で用いている `lp_solve` は，高性能ではないが数千変数程度まで使用可能なシステムであるため，まだ余裕があるものと考えられる．

これらの条件より，実用的な問題で実用的時間内である数時間から 1 日弱程度で解が得られる問題の規模は，計算時間がボトルネックとなり，ノード数に関しては百数十ノードであると考えられる．

6.6 当番表への応用

6.6.1 スケジューリング問題

ある作業を毎日繰り返して実行しなければならないことはよくある．通常このような作業は複数人の人間で分担して実行しているが，その人毎の予定や希望などが存在するため，単純に割った日数について作業して

もらうのは困難である．

また作業内容によっては連続して作業を行うことが不可能だったりする場合や，同時に行わなければならない作業が複数存在する場合がよくある．

このような割り当て問題はスケジューリング問題と呼ばれ，作業一つ一つをノードとし誰が行うかをノードの要素とすることで，COPにより表現できる．また個別に特別なノードを設定することにより，一人当りの作業時間数の合計や，作業間隔の指定なども可能である．

通常このようなCOPを形成した場合，最低限の取り決め(作業回数の分担など)のみ反映したCOPを形成すれば大抵の場合解が存在する．しかし，個人の希望や作業間隔などをも含めて制約に反映したCOPを形成すると往々にして過制約となり，解が得られない場合が多くある．このように個人の希望や作業間隔などのようなある程度妥協可能な制約についてはsoft制約とし，問題をPCOPとして解くことにより，なるべく多くの事項に関して満たされるようにすることが可能である．実際の応用としては24時間勤務の勤務交代を含む仕事の割り当て表や授業の割り当て表などが存在する．

6.6.2 RFLG法

スケジューリング問題の解法は様々なものがあげられるが，その一つとしてRFLG(Really Full Lookahead Greedy)法[19, 25]がある．これは元々は高校の授業の割り当て問題を解くためにつくられた解法であり，スケジューリング問題に対する有用な解法の一つである．

RFLG 法は基本的には状態空間型アルゴリズムであり，GSAT との違いはその初期点の決め方にある．GSAT では初期点にランダムに決定される点を用いるのに対し，RFLG 法は木探索法同様に制約を満たすように解を順に割り当て，割り当てられなくなったら解コストが最小になるように解を割り当てた点を用いる．

これにより初期点がランダムに与えられる GSAT に比較すると，初期点は解コストが低く実行可能解に近い点となる．そのため山登り探索の結果得られる解コストも低くなり，解にたどり着くまでの山登り探索の回数も減るため計算時間も短くなる．

6.6.3 スケジューリング問題による比較

RS 法の実際問題への適用例としてスケジューリング問題を用いた．比較の解法としては GSAT 及び RFLG 法を用いた．

スケジュールの内容は以下の通りとした．

- 15 人・一月 (30 日) 分の処理を行う．
- 毎日必ず誰かが□と◇の作業を行う．
- 一人が一日当たる作業は□か◇かとし，同じ日に同じ人が□と◇の作業が当たらないようにする．
- 一人一人決まった回数 (4 回) の作業を行う．
- なるべく二日続けて作業を行わないようにする．
- なるべく個人の希望日に作業が当たるようにする．

実際のPCOPは□と◇のノードを日数分だけつくり，それぞれのノードの要素として15人分としてAからOまでをとった．ノードが作業を表し，ノードの要素が作業を行う人を表している．これ以外に特定のノードを設けることにより，先にあげた条件を満たすようにした．なるべく二日続けて作業を行わないようにする条件となるべく個人の希望日に作業が当たるようにする条件をsoft制約とし，それ以外はhard制約として解いた．

PCOPの問題の規模はノード数60・ノードの要素数15・アーク密度0.0825となった．希望日の密度を変えることによりsoft制約の割合を変化させ3問生成し，解コストと計算時間を求めた．GSAT,RFLG法についてはそのまま解き，RS法についてはノードに基づく帰着法を用いて解いた．

一人一人決まった回数(4回)の作業を行う制約についてはそれぞれの回数毎によるノードを設けることにより記述可能である．しかしすべての組合わせを列挙するとノードの要素数が増える．そこで複数のノードをまとめそれぞれに対しノードを割り当て，さらに割り当てられたノードに対してさらにノードを割り当てることにより，CSPにより記述可能である．

ここで用いた問題では，一人一人決まった回数(4回)の作業を行う制約はhard制約であるため，ノードを用いた記述を行う必要はなく，直接制約式の記述を行うことが可能である．そこでRS法についてはILPの制約式として記述することにより問題の簡素化を行った．またGSAT及びRFLG法についても常に満たす制約として記述することにより，問題の簡素化を行った．

	soft 制約比	RS cost	GSAT cost	RFLG cost
問題 1	0.45	80	1042	60
問題 2	0.25	60	1139	163
問題 3	0.15	260	1076	383

表 6.10: スケジューリング問題による解コストの変化

	soft 制約比	RS time	GSAT time	RFLG time
問題 1	0.45	29.58	13215.43	1101.87
問題 2	0.25	17.31	9622.95	1099.98
問題 3	0.15	2.04	15668.56	1070.53

表 6.11: スケジューリング問題による計算時間の変化 (時間の単位は [sec])

ノードの重みは希望日となれば 1 とし，それ以外では 1 から 100 となるようにランダムに決定し，アークの重みは同じ作業が連続する場合は 20 とし，異なる作業が連続する場合は 10 とした．計算は PentiumII350MHz の PC 上で行った．

結果を表 6.10，表 6.11 に示す．

解コストについては問題 2 では最適解が得られ，問題 1，問題 3 でもかなり良い解が得られているのに対し，状態空間型アルゴリズムである GSAT はいずれも局所最適に陥っている．またスケジューリング問題に対して有効な解法であるとされる RFLG 法では GSAT に比較すると良い解が得られているものの，RS 法の方が良い解が得られている．

実際のスケジューリング問題は，問題にも依存するが，大抵は制約が緩い問題である．ここで用いた問題も，解構成型アルゴリズム・状態空

間型アルゴリズムとの比較評価で用いた問題と比較すると，制約が緩い問題でありRS法に適している．

RS法がGSATに比較して良い解が得られているのは制約が緩いためと考えられる．ここで用いた問題は，状態空間型アルゴリズムとの比較で用いた問題に対して，ノード数が多くあるのに対しアークの数が少なく探索空間が広く局所最適が数多く存在するためと考えられる．RS法がRFLG法に比較して良い解が得られているのは単体法により実数最適解を初期点として探索を行っているためと考えられる．RFLG法は初期点に解コストが低い点を用いているが，局所的な探索に基づいた点である．そのためGSATに比較すると良い解が得られるものの，単体法を用いることで解空間全体を対象とした探索を行うRS法より解コストが高くなりやすいと考えられる．

計算時間に関しては，ノード数が大きく問題規模が大きいため，山登り探索に要する時間が大部分を占める．RS法では単体法の反復適用で得られている解が良い解であり，山登りをほとんど必要とせず，計算時間は短い．これに対し，GSATでは初期点がランダムに決定されるため山登りの終了までの距離が長くなり，山登り探索に要する時間が長くなる．RFLG法は初期点が比較的良い点であるため，山登りの回数が少なくなるが，RS法と比較すると長くなる．

問題2について実際に得られた解を表6.12，表6.13，表6.14に示す．GSATによる解ではOの16,17及びJの19,20で同じ作業が連続し，Eの3,4及びAの17,18で異なる作業が連続する．またRFLG法による解ではJの8,9，Hの9,10及びNの17,18で異なる作業が連続する．これに対し，RS法による解では連続して当たることなく解が得られており，良い解が

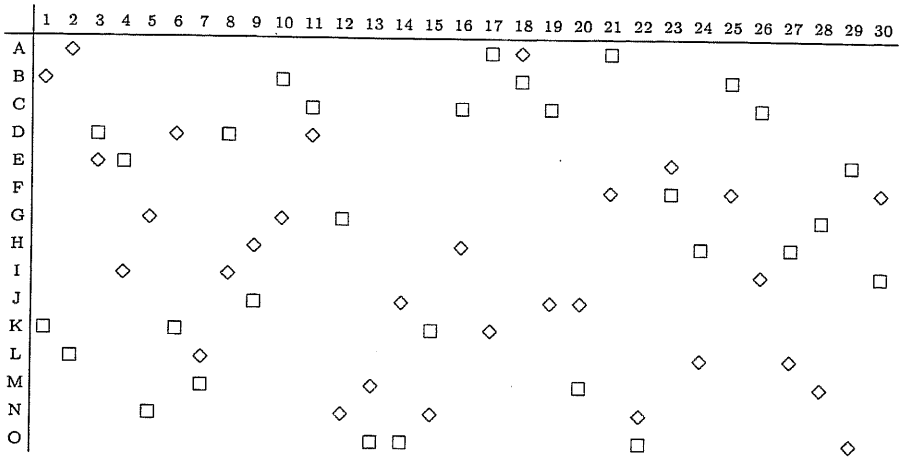

表 6.12: GSAT によるスケジューリング問題の解

得られていることがわかる.

6.6.4 実際の問題

表 6.15 は，実際に 2 月の仕事の割り当てについて PCOP を用いて処理した例である．この例は一日に二人がそれぞれ□と◇の作業を行うものであり，基本的には先にあげたスケジューリング問題と同様の問題である．先にあげたスケジューリング問題との違いは，各自の制約がきつくなっており作業に当たれる日数が 15 日程度しかない点と各自の規定回数が人によって異なる点である．違いのうち前者に関しては，制約がきつくなるため，スケジューリング問題としては困難な問題となる.

実際の計算は PentiumII350MHz の PC 上で行っており，計算に要する時間は十数分である．結果は C が 20,21 と作業をし，D が 10,11 と作業をすることになったが，全員の希望をかなえつつ大部分について連続して作業が当たらないようになっている.

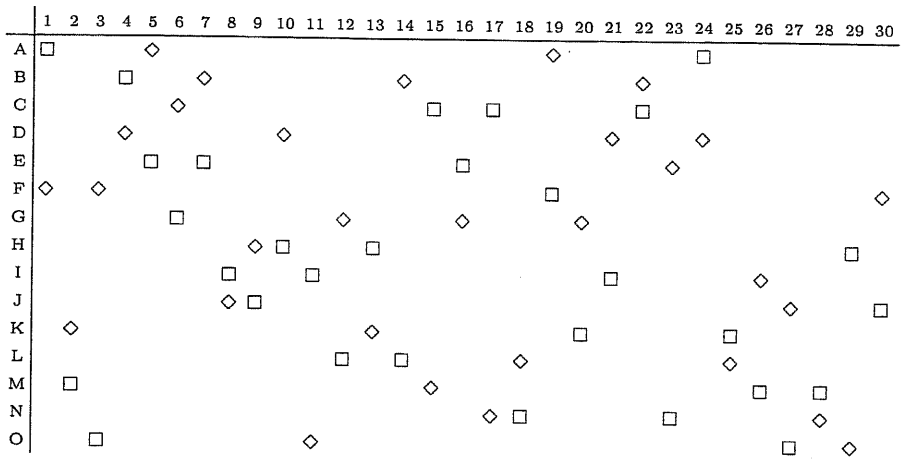

表 6.13: RFLG 法によるスケジューリング問題の解

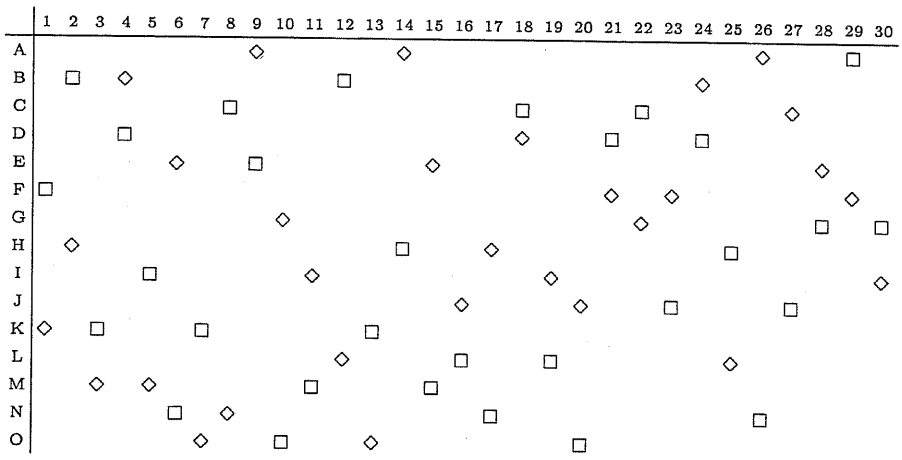

表 6.14: RS 法によるスケジューリング問題の解

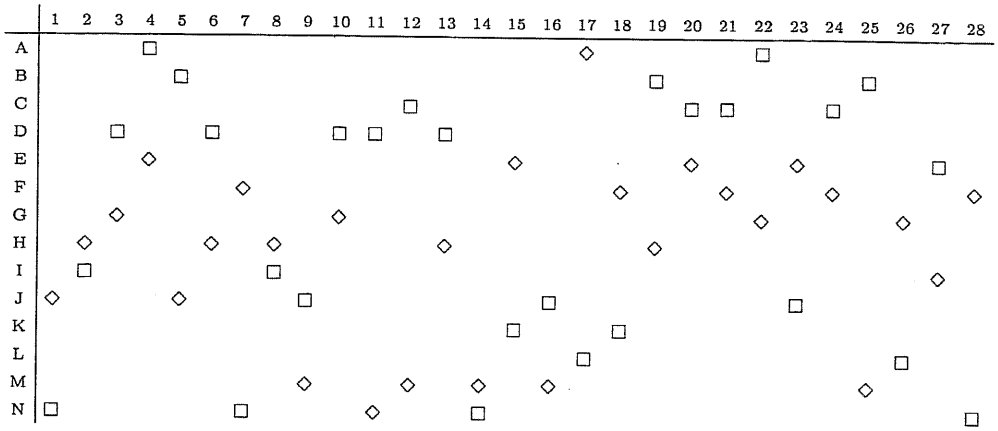

表 6.15: 当番表の割り当て例

従来人手でやっていたものを計算機上で解くようにしたとき，なるべく二日続けて作業を行わないようにする条件を外して行うと，二日続けて作業を行うことが多発した．また二日続けて作業を行わない条件をそのまま使用すると，制約過多により解が得られなかった．本手法を用いることで，制約過多の問題を回避しつつ良い解が実用的な時間以内で得られており，十分実用的であることが示されている．

6.7 その他の応用

PCOP の応用範囲は広くその他の応用としては以下のようなものがあげられる．

6.7.1 三面図理解

三面図は 3 次元物体の 2 次元平面上での表現法として，機械製図などで歴史的に多く用いられている．三面図は平面図であるため 3 次元物体

そのものを復元するには三面図の解釈という行為が必要になる．

現在の三面図理解システムは入力図面には誤りの無いと仮定したシステムしかなく，実用的なシステムはほとんど存在しない．また板金に限定するなど制限を設ける事により，習慣的に省略される隠れ線などを補うことで，図法上の誤りを含んだものでも3次元モデルが起こせるものもある．しかしこのようなシステムは習慣的な誤りを補うだけであり，全く関係ない誤りについて修正を行うことは不可能である．また用途も限定されており，部分的に誤りを含む三面図を処理できる汎用的な三面図理解システムは存在しない．

特に古い図面は電子化されていないため，三面図の取り込みはスキャナーなどを用いることになり，得られた情報そのものの正確さが保証されない．通常図面上に人間が表現する際に誤りが入る確率は非常に低いと考えられるので，このようなシステムの需要はそれほど多くはないと考えられるが，読み込んだ図面などでは線の擦れなどにより，誤り率が高くなっていると考えられる．

例としては図6.13があげられる．図6.13の右側が三面図である．三面図中の点線が存在する場合は左側の下の立体が三面図から生成される立体の一つとなる．しかしながら点線が擦れなどにより読み込まれず，ないものとされると，左側の上の立体が三面図から生成された立体の一つとなる．この例は単純な例であるが，実際にはより複雑な形状をしているため，擦れなどによる誤りは多くなるものと思われる．

三面図についてはその構造をCSPとして記述する方法は完成しており[8]，あとは解法の問題である．この点に関して，PCOPとして解くことにより，ある程度誤りを含む三面図についても図面理解が行えるように

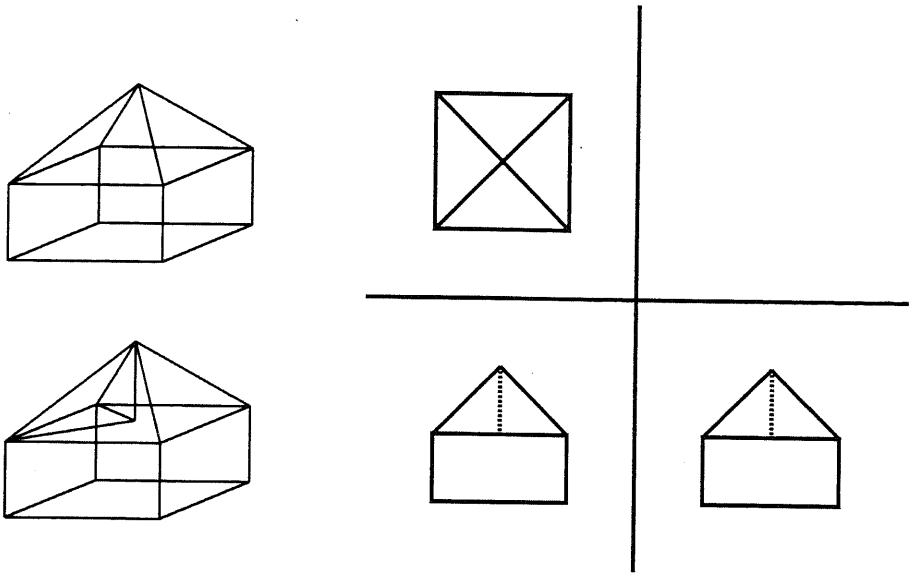

図 6.13: 三面図の例

なる．三面図では多くの線・面・頂点が存在しそれぞれについてノードが割り当てられるため，変数の数が多くなるためPBBの手法を用いることでは困難である．その点，本手法により高速化が行えるため，ある程度は実用化可能ではないかと考えられる．

特に本手法では制約の不成立に対して個別の重みがつけられる．そのため，線の掠れがあり定かではない線に関しては不成立の重みを低く，はっきりとした線に関しては不成立の重みを大きくすることにより，実用に即した三面図理解が行えるものと考えられる．

6.7.2 グラフの配置問題

グラフをいかに配置するかという問題であり，レイアウト問題に相当する [27]．CSPのようなグラフを配置するときに，配置のやり方により

そのグラフの構造の理解しやすさが大きく異なる．特にグラフの密度がある程度低い場合はノード間の接続が希薄となるため，アークが目立つようになり，無駄に交差しているアークや空間内のアークのばらつきが大きい場合は見た目も悪くなり，構造の把握も困難になる．

このような問題はアークが重ならないように制約を記述し，PCOPとすることによりうまく配置できると考えられる．この場合は制約が満たされないことによるコストはそれほど大きくはならず，ノードの配置のコストに対して数倍程度を設定することにより処理できるのではないかと考えられるため，特に本手法が有効に働くのではないかと考えられる．

6.7.3 診断システム

多くの診断システムはその診断方法とその診断の基となる観測結果に関しては100%正しいものとして処理を行っている．多くの場合はこのような前提は正しいと言えるが，生化学計測のように観測結果そのものが間違っている可能性や，診断の基となる診断方法が間違っている可能性も存在する．

このような場合，従来のCOPを用いた診断システムでは，無理やりすべての解を満たすような診断結果を出すか，エラーであるとして，診断そのものが全く出力されない．

このような問題に対してPCOPを用いて解くことにより，大部分の観測結果や診断方法にのっとりながら総合的に正しいと考えられる診断が得られるようになる．また誤りとなった観測結果や診断方法も判明するため，再度観測を行ったり，診断方法を検討することができ，的確な診

断が行えるようになる.

第7章　まとめ

7.1　結語

本稿ではCSPの拡張であるPCOPについて，その高速解法の手法と実装について示した．

スケジューリング問題など現実の問題の多くはCSPによって記述する事で効率的に解が得られるが，実際には相反する条件が存在するなどCSPの範疇では処理不可能なものが数多くあった．これに対しPCOPを用いる事で相反する条件も含めあらゆる条件が記述できるため，応用範囲が大きく広まるものと考えられる．

PCOPを用いる際に最も大きな問題となるが解を得るまでの時間である．用途によっては時間がかけられる場合もあるが，対話システムのように時間が限られる場合も数多くある．この点でPCOPの高速解法は有意義なものであると言えよう．また現実問題では規模が大きすぎるため，PBB等のような全解探索法では実用に耐えないことがほとんどであると考えられる．

本手法によるPCOPの解法は単体法を用いているため計算時間は問題規模の多項式オーダですむと考えられる．これは実際の問題を解く上で重要な性質である．すべての解を得る場合，本手法の限界はノード数にして百数十であるが，大規模の問題でない限り実用に足りる．実際に十

数人規模のスケジューリングでは十分実用的な処理法であり，利用可能である．また反復適用の途中結果に対して丸めを行った解も最適解に近い解であると考えられるため，解の質はやや劣るものの，より大規模な問題も解けるものと考えられる．

本手法は制約のきつい問題には適さない．これは他の解法を用いた方が，より短い計算時間で，同じかやや良い解が得られるためである．しかしながら，PCOP となりながら制約が緩い問題も存在する．特に従来 CSP として設定され解かれているスケジューリング問題などでは，緩い制約を付加しても PCSP となる場合が存在する．この場合問題そのものは PCSP ではあるが，制約が緩いため，本手法により解くことにより効率的に解が得られる．その意味でも本手法は有効である．

7.2 今後の課題

今後の課題としては以下の 2 点があげられる．

制約がきつい問題の解法 制約がきつい問題は本手法を用いても効率的に解けない．これは単体法により得られる実数解が最適解を反映したものとならないためである．これに対しては ILP への帰着法そのものを変更する必要があり，制約がきつい場合でも最適解を反映した実数解が得られる帰着法を探し出す必要がある．

計算方法の改善 現在の計算方法ではノード数が百数十が限界である．中規模の問題にはこれでも処理できるが，大規模の問題になると計算できなくなる．途中の解を用いることによりある程度まで拡張でき

るが，それでも限界がある．広く応用する上では，帰着の方法も含め改善を行うことにより，より大規模な問題も解けるようにする必要がある．

謝辞

本研究を進めるにあたり，多くの方々から御指導・御鞭撻を賜りました．

石塚満教授には大変ご多忙の身でありながら熱心に御指導していただきました．単体法の反復適用の考えは先生の御助言なしには考えられなかったと思います．感謝致します．

伊庭助教授には遺伝的アルゴリズムとの比較について教えて頂くなど様々に御指導頂きました．感謝致します．

土肥助手にはプログラムの実装について教えて頂くなど様々に御指導頂きました．感謝致します．

蔡東風氏には制約最適化問題の 0–1 整数計画法の置き換え手法について教えて頂くなど様々に御指導いただきました．感謝致します．

松尾豊氏には単体法の反復適用の手法について相談にのってもらうなど様々に助けてもらいました．感謝致します．

研究室のメンバーの皆さんには，本論文の研究に留まらず研究室での生活など様々な面でサポートしていただきました．特に先輩である高間氏には人工知能全般に関して教えて頂きました．感謝致します．

医学部医用電子および先端研人工生体計測の皆さんには，開発環境およびデータ収集に協力していただいた他，様々にサポートしていただき

ました．特に不完全制約最適化問題に取り組む動機は，ここでの実験で得ました．感謝致します．

最後に経済面・生活面・精神面において私を支えてもらいました両親に感謝致します．

参考文献

[1] 今野 浩: 数理計画法, 産業図書, 1981.

[2] Eugene C. Freuder, Richard J. Wallace: Patial Constraint Satisfaction, *Artifical Intelligence*, Vol.58, pp.63–110, 1992.

[3] Steven Minton, Mark D. Johnston, Andrew B. Philips, Philip Laird: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, *Artifical Intelligence*, Vol.58, pp.161–205, 1992.

[4] Vipin Kumar: Algorithms for Constraint – Satisfaction Problems: A Survey, *AI Magazine*, Vol.13, No.1, pp.32–44, 1992.

[5] 茨木 俊秀, 福島 雅夫: 最適化の手法, 共立出版, 1993.

[6] E.Tsang: Foundations of Constraint Satisfaction, *Academic Press*, 1993.

[7] R. R. Bakker, F. Dikker, F. Tempelman, P. M. Wongnum: Diagnosing and solving over-determined constraint satisfaction problems, *IJCAI'93*, pp.276–281, 1993.

[8] 西原 清一: 図面理解による3次元形状モデリング, *Computer Today*, No.56, pp.19–29 (July 1993).

[9] Javier Larrosa, Pedro Meseguer: Optimization – based Heuristics for Maximal Constraint Satisfaction, *Constraint Satiscfaction, CP'95*, pp.103–120, 1995.

[10] Nils Lenke: Natural Language Generation as Constraint-Based Configuration, *Lecture Notes in Artifical Intelligence*, No.992, pp.1–12, 1995.

[11] Pedro Meseguer, Javier Larrosa: Constraint Satisfaction as Global Optimization, *IJCAI'95*, pp.579–584, 1995.

[12] Yves Caseau, François Laburthe: Improving Branch and Bound for Jobshop Scheduling with Constraint Propagation, *CCS'95*, pp.129–149, 1995.

[13] Peter Barth: Logic-Based 0–1 Constraint Programming, *Kluwer Academic Publishers*, 1996.

[14] KenMcAloon and Carol Tretkoff: Optimization and Computational Logic, *Wiley-Interscience Publication*, 1996.

[15] Hoong Chuin LAU: A New Approach for Weighted Constraint Satisfaction: Theoretical and Computational Results, *Constraint Satisfaction, CP'96*, pp.323–337, 1996.

[16] Michael Jampel: A Brief Overview of Over-Constrained Systems, *Lecture Notes in Computer Science*, No. 1106, 1996.

[17] Richard J. Wallace: Enhancements of Branch and Bound Methods for the Maximal Constraint Satisfaction Problem, *AAAI'96*, pp.188–195, 1996.

[18] Walter Hower and Winfried H. Graf: A bibliographical survey of constraint-based approaches to CAD, graphics, layout, visualization, and related topics, *Knowledge-Based System*, Vol.9, No.7, pp.449–464, 1996.

[19] Masazumi Yoshikawa, Kazuya Kaneko, Toru Yamanouchi and Masunobu Watanabe: A Constraint–Based High School Scheduling System, *IEEE Expert Intelligent Systems*, Vol.11, No.1, pp.63–72, 1996.

[20] 石塚 満: 知識の表現と高速推論, 丸善, 1996.

[21] Pascal Van Hentenryck, Laurent Michel and Yves Deville: Numerica A Modeling Language for Global Optimization, *The MIT Press*, 1997.

[22] 特集:「制約充足問題の基礎と応用」, 人工知能学会誌, Vol.12, No.3, pp.350–389, 1997.

[23] Kim Marriott and Peter J. Stuckey: Programming with Constraints: An Introduction, *The MIT Press*, 1998.

[24] M.J.Atallah (ed.): Algorithms and Theory of Computation Handbook, *CRC Press*, 1999.

[25] Siu Cheung Kong and Lam For Kwok: A conseptual model of knowledge–based time–tabling system, *Knowledge-Based Systems*, Vol.12, No.3, 1999.

[26] 蔡 東風, 石塚 満: 線形計画法を利用した離散制約最適化問題の効率的近似解法人工知能学会誌, Vol.14, No.2, pp.334–341, 1999.

[27] 中野 眞一, 西崎 隆夫: グラフの自動描画, 電子情報通信学会誌, Vol.82, No.2, pp.175–180, 1999.

[28] `lp_solve`(Ver.2.3), `ftp://ftp.es.ele.tue.nl/pub/lp_solve/`

[29] `ruby`(Ver.1.4), `http://www.ruby-lang.org/`

発表文献

[1] 斎藤 逸郎, 正木 寛人, 奥乃 博, 石塚 満: 二分決定グラフによる三面図理解システムの機能拡張, 第 50 回情報処理学会 全大, No.2, pp.409–410, March, 1995.

[2] 斎藤 逸郎, 正木 寛人, 石塚 満, 奥乃 博: 二分決定グラフを用いた三面図理解システム, 第 9 回人工知能学会 全大, pp.557–560, July, 1995.

[3] 斎藤 逸郎, 蔡 東風, 石塚 満: CSP におけるノードの値の重みとアーク重みの相互変換, 第 56 回情報処理学会 全大, No.2, pp.6–7, March, 1998.

[4] 斎藤 逸郎, 蔡 東風, 石塚 満: 制約充足問題におけるノードの値の重みとアーク重みの相互変換, 第 12 回人工知能学会 全大, pp.322–323, June, 1998.

[5] 斎藤 逸郎, 石塚 満: 数理計画法を用いた不完全 CSP の高速近似解法, 第 58 回情報処理学会 全大, No.2, pp.233–234, March, 1999.

[6] 斎藤 逸郎, 石塚 満: 単体法の反復適用を用いた不完全 CSP の高速近似解法, 第 13 回人工知能学会 全大, pp.9–11, June, 1999.

[7] 斎藤 逸郎, 石塚 満: 単体法の反復適用による不完全CSPの高速近似解法, 第59回情報処理学会 全大, No.2, pp.55–56, September, 1999.

[8] 斎藤 逸郎, 石塚 満: 制約充足問題におけるノードの値の重みとアーク重みの相互変換, 人工知能学会誌, Vol.15, No.3, 2000. (掲載予定)

本博士論文の内容外の発表文献

[1] 正木 寛人, 斎藤 逸郎, 石塚 満, 奥之 博: 二分決定グラフの適用による三面図の効率的理解, 情報処理学会論文誌, Vol.37, pp.1969–1979, 1996.

[2] 斎藤 逸郎, 土肥 浩, 石塚 満: WWWにおけるグループ経験の共有を図るメディエータエージェントの構築, 第53回情報処理学会 全大, No.4, pp.239–240, September, 1996.

[3] 斎藤 逸郎, 山田 仁, 石塚 満: WWWにおけるグループ内での経験の共有を行うメディエータエージェントの実装, 第54回情報処理学会 全大, No.3, pp.143–144, March, 1997.

[4] 斎藤 逸郎, 石塚 満: WWWアクセス経験のグループ共有を行うメディエータエージェントのユーザインターフェースの改良, 第55回情報処理学会 全大, No.3, pp.180–181, September, 1997.

[5] I. Saito, T. Chinzei, S. Mochizuki, Y. Abe, T. Suzuki, T. Karita, T. Ono, A. Kouno, K. Baba, K. Imachi: The Control Method of The Undulation Pump Total Artificial Heart, XXV European Society for Artifical Organs, pp.644, November, 1998.