Vol. 33 No. 5 情報処理学会論文誌 May 1992

制約違反の十分条件を用いた組合せ制約充足/最適化†

丸山文宏†† 箕田依子†† 澤田秀穂††
滝沢ユカ†† 川戸信明††

組合せ的な制約充足や最適化は，理論的にその困難さがわかっている（NP 困難）問題であるが，工学や OR の様々な局面に現れる重要な問題である．本研究の目的は，これらのうち制約条件が不等式および等式で表現できる問題に対して，できるだけ効率のよい解法を開発することである．本論文では，このような問題に対する厳密解法を考察し，新しい制約充足アルゴリズムおよびこれに基づく最適化手法を提案する．本手法では，制約充足処理の失敗情報として，違反の根拠を示す，制約違反の十分条件を論理式（不等式またはその論理積）で表現した制約違反条件を導入し，これを最大限に利用して制約充足/最適化を行う．その特徴は，①制約違反を検出すると制約違反条件を生成・蓄積して制約を伝播し，これにより未探査の解空間を削り，②制約充足を繰り返すことにより最適化を行い，③制約条件や評価関数が非線形の場合にも適用できる，ことである．実験システムを実装し例題で評価した結果，制約違反条件が解空間の絞り込みに有効であること，他の手法と比べて 10 倍以上高速に最適化できること，例題に関して十分実用になる性能を持つこと，を確認した．さらに規模の大きな問題に対しても，この手法が近似解法を開発するベースになりうる．

1. はじめに

組合せ的な制約充足や最適化は，工学や OR の様々な局面に現れる重要な問題であるが，潜在的な解空間が組合せ的に大きくなり，その困難さが理論的にわかっている（NP 困難）．多くの問題では，厳密な最適解を求めることができるのは比較的小規模な問題に限られ，実用的な規模の問題に対しては近似解法が実際上唯一の手段となっているのが現状である．

一方，解空間の探索を 1 つの出発点とした AI 技術も，バックトラックによる単純な探索技術から，依存関係を利用した探索制御，整合性維持による解空間の削減，解空間内における制約伝播[1] という手法を開発している．現在では，このような考え方を利用して実用的な組合せ問題の効率よい解法を開発し，工学や OR に貢献できる可能性がある．

本論文では，以下のように定式化できる重要な問題群に注目し，その厳密解法を考察する．

『n 個の変数 $x_1, x_2, \cdots, x_n$ について，不等式系

$$\sum_{i=1}^{n} a_{ki} d_k(x_i) \leqq b_k \quad (k=1, 2, \cdots, m) \qquad (1)$$

と評価関数

$$\sum_{i=1}^{n} a_{0i} d_0(x_i) \qquad (2)$$

が与えられており（制約充足では（2）式がない），変数 x_i の取りうる値は離散的な c_{ij} $(j=1, 2, 3, \cdots)$ であるとする．ただし，変数値 c_{ij} はそれ自身が具体的な数値ではなく，いくつかの属性値 $d_k(c_{ij})<k=0, 1, 2, \cdots, m>$ を持つ複合値である．この時，（1）式の不等式をすべて満たしかつ（2）式の評価関数を最小（または最大）にする（制約充足では（1）式の不等式をすべて満たすだけでよい）ような各変数の値を決定する』

不等式系（制約条件）および評価関数は上のように線形である必要はない．また，不等号の向きは逆でもよいし，等式が含まれていても構わない．非線形で等式が含まれる例を付録 1 に示す．等式は 2 つの不等式で表している．

上のように定式化できる問題には，ナップサック問題[2]，材木や紙の切断問題（cutting-stock problem)[3]，回路規模とスピードの制約条件のもとで行う LSI の論理設計[4),5] 等実用上重要な問題が含まれ，スケジューリング問題にも適用可能である．

例えば，LSI の論理設計については次のようにする．まず，回路中の各コンポーネントに対応して変数 $x_1, x_2, \cdots, x_n$ を設ける．b_k $(k=1, 2, \cdots, m)$ を k 番目のパス（パス k）に沿った遅延時間の制約値（上限），パス k $(k=1, 2, \cdots, m)$ が i 番目のコンポーネントを通っていれば $a_{ki}=1$，通っていなければ $a_{ki}=0$，$d_k(x_i)$ $(k=1, 2, \cdots, m)$ を i 番目のコンポーネント内のパス k に沿った遅延時間とすれば，（1）式は遅延時間の上限に関する制約を表現している．$d_0(x_i)$ を i 番目のコンポーネントのゲート数として評価関数を

$$\sum_{i=1}^{n} d_0(x_i)$$

† Constraint Satisfaction and Optimization Using Sufficient Conditions for Constraint Violation by FUMIHIRO MARUYAMA, YORIKO MINODA, SHUHO SAWADA, YUKA TAKIZAWA and NOBUAKI KAWATO (Artificial Intelligence Laboratory, Fujitsu Laboratories Ltd.).

†† (株)富士通研究所人工知能研究部

とすれば，遅延時間の制約のもとで回路を最小にすることになる．各変数 x_i の取りうる値 c_{ij} $(j=1,2,3,\cdots)$ は，この場合，各コンポーネントの実現方法である．

上の問題は，変数値 c_{ij} $(i=1,2,\cdots,n;\ j=1,2,3,\cdots)$ に対応した 0-1 変数 x_{ij} を設け $d_k(x_i)$ の代わりに

$$\sum_j d_k(c_{ij})x_{ij} \quad (d_k(c_{ij}) \text{ は定数}) \qquad (3)$$

とし，次の条件を付け加えることにより，整数計画問題として定式化できる．

$$\sum_j x_{ij}=1 \quad (i=1,2,\cdots,n),$$

$$0\leqq x_{ij}\leqq 1,\ x_{ij}\in Z \quad (i=1,2,\cdots,n;\ j=1,2,3,\cdots).$$

整数計画問題の解法（整数計画法）としては，まず，整数値を取るという条件を外した問題（緩和問題）を解き，その最適解に近い整数解を取るという方法がある．この時，緩和問題の最適解の成分の0でないものが比較的大きな値を持つ問題に対しては，四捨五入した整数解が元々の問題の最適解に関するよい情報を提供することが知られている．しかし，この方法は，値が狭い範囲に絞られている問題（例えば，上のような 0-1 変数問題）に対してはうまく機能しない[6]．

次に，分枝限定法[7]では，分枝（branching operation）によって生成した部分問題を順に解き，それまでに部分問題の最適解として得られた解のうち最良のものを暫定解として保持しておく．部分問題の緩和問題の最適解とこの暫定解を比較し，もし前者が後者より良くはないならば，この部分問題からは原問題の最適解は得られないとして限定（bounding operation）を行い，そこで分枝操作を打ち切る．このようにして探索木の枝刈りを行う．限定されていない部分問題がなくなれば，その時の暫定解が原問題の最適解である．現在のところ，分枝限定法が実用的な規模の整数計画問題を解く唯一の手法であると言える．

しかし，整数計画問題としての定式化は，変数の数が増えるという問題点を持つ．例えば，上で述べたような 0-1 変数の導入による定式化では，変数の数が変数値の平均個数倍に増える．整数計画法の計画時間は，変数の数について指数的に増加する傾向があるため，変数の増加により計算時間が大幅に増大する恐れがある．また，あらかじめすべての変数値とその属性値（(3)式の $d_k(c_{ij})$）がわかっていなければならず，必要に応じて列挙していくことはできない．

知的バックトラック（dependency-directed backtrack）[8]では，制約違反の解消に関係ない分岐点をスキップして，制約解消に関係のある分岐点でのみ選択肢の変更を行うが，制約を伝播するということは考えない．また，本論文で取り上げる切断問題のように制約条件も評価関数もすべての変数を含んでいる問題に対しては，すべての分岐点で選択肢の変更が有効である可能性があるため，分岐点がスキップできず，効率が悪い．

ATMS[9]では，$x_i=c_{ij}$ $(i=1,2,\cdots,n;\ j=1,2,3,\cdots)$ をそれぞれ仮説（assumption）として，これらを要素とする集合で矛盾（制約違反）を引き起こすもの（nogood）を削除していく．しかし，仮説の組合せの数が膨大となるため扱える規模が限られる．ATMSと本論文の手法との比較については付録2でも触れる．

Dincbas らは制約論理型言語 CHIP によるアプローチについて報告している[3]．その特徴は，各変数値（選択肢）に対応する変数を設けることなく，これらをまとめて1つの変数で扱うことにある．このため，変数値の領域（有限とする）を管理し，これを絞り込む機構を備えている．すなわち，変数間の等式や不等式から制約充足/最適化不能になることがわかる変数値を削除することにより，変数の値の範囲を狭めて，この制約を伝播する．また，制約充足を繰り返すことで最適化を行う．しかし，CHIP には ATMS のように複数の変数の変数値の組合せを禁止する手段はない．したがって，効率を上げるため，変数値の領域を二分していくことによる分枝限定（dichotomic search）を導入している．

本論文では，制約充足処理の失敗情報として，違反を引き起こす変数値の組合せではなく，違反の根拠を示す，制約違反の十分条件を論理式で表現した制約違反条件を導入し，これを最大限に利用した制約充足/最適化手法を提案する．その特徴は次のとおりである．

①制約違反を検出すると制約違反条件を生成・蓄積して制約を伝播し，これにより未探査の解空間を削る．

②対応する制約充足問題の解が求まると，蓄積している制約違反条件に含まれる評価関数の制約値をこの解の評価関数値を用いて更新し，このような制約充足を繰り返すことにより最適化を行う．

③制約条件や評価関数が非線形の場合にも適用できる．

2. 制約充足/最適化手法

本章では，本手法のキーである制約違反条件を定義し，これに基づく制約充足のアルゴリズムを提案し，

このアルゴリズムを利用した最適化について述べる．具体的な制約違反条件の形やアルゴリズムの動きについては，付録2の実行トレースを併せて参照されたい．

2.1 制約違反条件（NJ）

制約違反のための十分条件を論理式（不等式またはその論理積）で表現したものを制約違反条件（NJ：Nogood Justification）と呼ぶ．制約違反条件は次のように帰納的に定義される．

①(1)の各不等式の逆（つまり，不等号を“＞”に変えたもの）は NJ であり，「制約違反不等式」と呼ぶ．

②変数に変数値が設定されている状態において（値が不定の変数があっても構わない），1つの変数 x_i に注目してその値を変えた時どの値 c_{ij} に対しても少なくとも1つの NJ（1つを D_j とする）が成立する場合（$j=1,2,3,\cdots$），D_j に含まれる x_i の属性値に c_{ij} の属性値を代入した D_j' の論理積 $\bigwedge_j D_j'$ も NJ である．ただし，NJ に含まれる変数の値が不定の時はその NJ は不成立と見なす．（定義終）☆

本節の初めに述べたように，NJ は制約違反のための十分条件である．正確には，次の命題が成り立つ．

[命題 1] 任意の NJ が成立すると，その時点で値が不定の変数にどのように値を割り当てても少なくとも1つの制約違反不等式が成立する．　□

(証明) 制約違反不等式が制約違反の十分条件であることは明らかだから，定義②で D_j がすべて制約違反の十分条件であると仮定し，生成される NJ $\bigwedge_j D_j'$ も制約違反の十分条件となることを言えばよい．

今この NJ が成立すると，$D_j'(j=1,2,3,\cdots)$ もすべて成立する．D_j と D_j' の関係から，変数 x_i に値 c_{ij} を割り当てると D_j が成立する（$j=1,2,3,\cdots$）．したがって，x_i にどのように値を割り当てても少なくとも1つの D_j が成立することになり，仮定から D_j は制約違反の十分条件だから，残りの変数にどのように値を割り当てても少なくとも1つの制約違反不等式が成立する．　(証明終)

②のように1つの変数に注目して生成された NJ にはその変数はもはや含まれない．これら，その変数を含む領域に対する制約がその変数を除いた残りの領域に対して伝播されたことになる．生成された直後の NJ はやはり成立しており，それを成立させないようにすることが次の目標となる．

②のように NJ が生成されるケースには次の2通りがある．まず，既に NJ がどれか成立しており，値が割り当てられている変数を1つ選んでその値を現在のものから他のものに変更する場合である．この操作を変数値変更と呼ぶ．次に，どの NJ も成立していないが，値が不定の変数が残っており（つまり，まだ解が得られていない），そのうちの1つを選んでその値を割り当てる場合である．この操作を変数値割当と呼ぶ．

2.2 制約充足アルゴリズム

制約違反条件（NJ）を用いて(1)の制約充足を行うアルゴリズムを以下に示す．

[手順 0] すべての変数に初期値を割り当てる．

[手順 1] 成立する NJ があれば手順2へ，なければ手順3へ．

[手順 2] 値が割り当てられている変数を1つ選び，変数値変更を行う．どの NJ も成立させない変数値があれば手順3へ．なければ，2.1 節②のように新しい NJ を生成し，この変数の値を不定として手順5へ．

[手順 3] 値が不定の変数が残っていれば手順4へ．残っていなければ，制約充足は成功で終了する．

[手順 4] 値が不定の変数を1つ選び，変数値割当を行う．どの NJ も成立させない変数値を割り当てられれば手順3へ．すべての変数値が NJ を成立させるなら，2.1 節②のように新しい NJ を生成し，この変数の値を不定のままとして手順5へ．

[手順 5] 生成された NJ に変数が含まれていれば手順2へ．含まれていないなら，制約充足は失敗で終了する．　(制約充足アルゴリズム終)

手順4では，最も最近値を不定とした変数を選ぶ．手順2における変数の選択法については，あらかじめ決めた順番に従って変数を選んでいく固定順方式とその都度何らかの尺度（例えば，属性値が最大）で変数を選ぶ変動順方式がある．

以下では，固定順方式による上記アルゴリズムの正当性を示す．まず，固定順方式では，深さ n の探索木を辿るアルゴリズムになることに注意する．例えば順番を $x_1, x_2, \cdots, x_n$ とした場合，根に近いレベルから $x_n, x_{n-1}, \cdots, x_1$ に対応する木を辿る（例えば，付録2図5）．すなわち，手順0である葉から出発し，手順2は兄弟を，手順4は子をそれぞれ探索することに相当する．手順2に成功すると兄弟に移り，失敗すると，

☆ 制約違反条件を考える代わりに，与えられた不等式を含むような，制約充足の必要条件を考え，論理積の代わりに制約充足の必要条件の論理和を考えても，本論文の議論は双対に進む．

NJ を生成して親に上り，それ以下の部分木を枝刈りする．手順4に成功すると子に降り，失敗すると，NJ を生成してこのノード以下の部分木を枝刈りする．手順3は葉にいるかどうかの判定である．

[補題] 探索木の各ノードは高々2回しか訪れない． □

(証明) まず，NJ が成立するノードから親または兄弟に移ると再びそのノードを訪れることはないことに注意する．あるノードを初めて訪れた時を考える．処理は変数値変更か割当である．変更なら，このノードで成立する NJ があり，次に兄弟か親に移るから，再びこのノードを訪れることはない．割当なら，手順4に失敗すれば NJ が生成され，手順5で終了するか次の手順2で兄弟または親に移るから，再びこのノードを訪れることはない．手順4に成功して子に降りると，再び訪れるのは手順2で失敗して NJ が生成された直後である．生成された NJ はこのノードで成立し，引き続く変数値変更で兄弟または親に移るから，再びこのノードを訪れることはない． (証明終)

[命題 2] 変数値が有限個ならば，2.2 節の制約充足アルゴリズムは正当である．すなわち，解が存在すれば対応する葉が停止し，解が存在しなければ変数を含まない NJ を生成して停止する． □

(証明) 変数値が有限個ならば探索木のノードは有限個であり，補題から各ノードは高々2回しか訪れないから，手順3または手順5で停止する．手順3で停止すれば，葉にいて（すべての変数の値が決まっている）しかもどの NJ も（したがってどの制約違反不等式も）成り立たない．したがって，(1)式の不等式をすべて満たす解が得られている．手順5で停止すれば，変数を含まない NJ が生成されるが，命題1によって，どのように変数値を割り当てても少なくとも1つの制約違反不等式が成り立つ．つまり，解は存在しない． (証明終)

このアルゴリズムの特長は，生成した NJ がまだ訪れない部分木の枝刈りをすることにある．この際，その子孫の葉が制約を満たさないことは，NJ が制約違反の十分条件であることから保証されるのである．

2.3 最適化

(1)と(2)式の最適化も制約充足を繰り返すことにより実行できる．まず，(2)式の評価関数に対応する不等式

$$\sum_{i=1}^{n} a_{0i} d_0(x_i) \leqq b_0 \qquad (4)$$

を設ける（これは(2)式を最小にする場合．最大にする場合は不等号を“≧”とする）．与えられた不等式系(1)に(4)式を付加した組合せ制約充足問題に対して，制約値 b_0 を適当な初期値に設定して 2.2 節のアルゴリズムを適用する．解が得られる度に b_0 を

得られた評価関数値$-\varepsilon^{\star}$ (ε は十分小さな正数)に更新して手順1以下を再び適用することを繰り返す．最終的に制約充足に失敗したら，その直前に得られた解が最適解である．この際，以前（制約値が現在の値になる前）に生成した NJ も参照しながら制約充足を行う．したがって，制約値が緩やかな時に制約充足に失敗した際の情報によって無駄な探索を省くことができる．

3. 実装と評価

3.1 実装の概要

本手法の評価を行うため，**図1**に示すような構成の実験システムをC言語で実装した．制約充足モジュールは 2.2 節のアルゴリズムを実現したものである．制御モジュールは最適化における繰り返しを始めとする制御を行う．入出力インタフェースは，不等式系(1)と評価関数(2)（具体的には，係数 a と属性値番号 k）の入力，制約値（b）の設定／変更，および結果（統計情報を含む）の出力を行う．制約違反条件は，実行開始時には制約違反不等式だけだが，実行が進むにつれ順次生成・蓄積される．変数値情報は各変数値の属性値であり，現在はあらかじめ配列に値を設定しておく．

この実験システムでは，入力された不等式系および評価関数の形を固定したまま，ユーザが制約値を変更する度に制約充足アルゴリズムの手順1から再実行することが可能である．この際，制約違反不等式以外の NJ を削除するかすべて残したままにするかの選択ができる．後者の場合は，制約値を変更しても対応する NJ はそのまま利用できる．この実験システムの規模

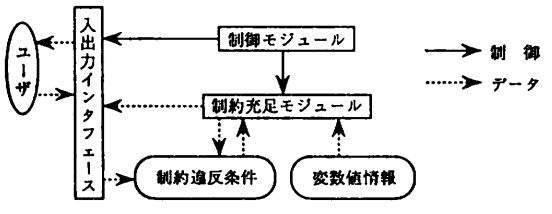


図 1 実験システムの構成
Fig. 1 Configuration of the experimental system.

☆ 評価関数値が整数に限られているなら ε は1にすればよい．また，(2)式を最大にする場合は“$+\varepsilon$”とする．

は，制御モジュールが約100行，制約充足モジュールが約400行であり，入出力インタフェースを除く本体は合わせて約500行である．

3.3 節の評価結果はすべて固定順方式 ($x_1, x_2, \cdots, x_n$) によるものである．固定順方式において実施した高速化のための改良には次のものがある．まず，生成される NJ を互いに素な集合 G_1〜G_{n-1} に分けることにする．ここで，G_i は変数 x_1〜x_i を含まない NJ の集合である ($i=1, 2, \cdots, n-1$)．変数 x_1〜x_i が不定の時検査をするには，すべての NJ をチェックする必要はなく，G_i に属する NJ だけでよい．

また，次に説明する切断問題のように対称な問題に対しては，1つの変数 (x_n) だけを含む NJ が生成されるとそれを成立させる変数値をすべての変数について禁止することにより高速化を図っている．

3.2 例　　題

評価用の例題としては文献 3) の材木の切断問題を用いた．**図2**に示すような大きな板から6種類の棚を切り出す際，使えない無駄な部分 (waste) を最小にする問題である．ただし，それぞれの棚の必要枚数は与えられている（これより多く切り出して構わない）．板は同じ枚数ずつのロットに分けられ，1つのロットでは皆同じ切り方をする．切断機は，同時に切ることができる刃の組を水平方向に1組，垂直方向に2組持っている．切り方は，まず，水平方向に切り細長い板にする．次に，細長い板を垂直方向に2組の刃のうちどちらかで切って棚を得る．**図3**に切り方の例を示す．斜線を施したのが使えない無駄な部分である．板は多くの切り方が考えられるが，1回の切断で生じる無駄の面積および同じ幅の棚の枚数に上限が設定され，切り方の総数が抑えられている（文献3）では

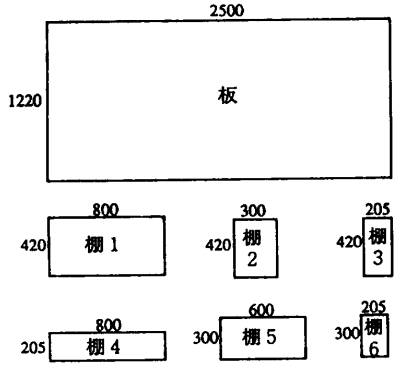


図 2 切断問題における板と棚
Fig. 2 Board and shelves in the cutting stock problem.

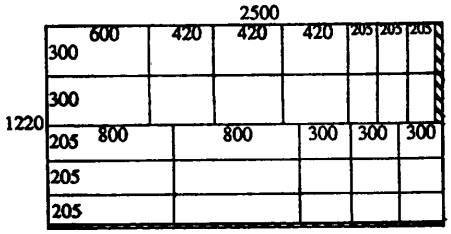


図 3 切り方の例
Fig. 3 An example of configuration.

表 1 棚の必要数
Table 1 Required number for each shelf.

棚	b_1	b_2	b_3	b_4	b_5	b_6
必要数	4	16	4	12	16	32

72 通り)．

定式化では，まず，各ロットに対応して切り方を値とする変数を設ける．評価関数は無駄の面積の総和である．結局，以下のようになる．

$$d_1(x_1)+\cdots+d_1(x_n)\geqq b_1,$$
$$d_2(x_1)+\cdots+d_2(x_n)\geqq b_2,$$
$$d_3(x_1)+\cdots+d_3(x_n)\geqq b_3,$$
$$d_4(x_1)+\cdots+d_4(x_n)\geqq b_4,$$
$$d_5(x_1)+\cdots+d_5(x_n)\geqq b_5,$$
$$d_6(x_1)+\cdots+d_6(x_n)\geqq b_6,$$
$$d_0(x_1)+\cdots+d_0(x_n)\rightarrow \text{最小}.$$

ただし，n はロット数，b_1〜b_6 はそれぞれの棚の必要枚数，d は属性値で，d_0 がロット全体の無駄の面積，d_1〜d_6 がロット全体でそれぞれの棚の取れる枚数である．b_1〜b_6 の値 ($n=4$ の場合) は**表1**に示す．ここでは，各ロットの板は1枚ずつとしている．

3.3 評 価 結 果

まず，n を4として最適化を行った．各ロットの切り方は文献 3) では 72 通りであるが，ここでは，72 通りの場合とさらに 20 通りの切り方を考慮した 92 通りの場合を実験した．後者の実行時間は前者とほとんど変わらず，後者の解（無駄面積 1.87%）は前者（無駄面積 4.14%）より改善された．

次に，n を増やして実験を行った．切り方は 92 通りである．制約値 (b_1〜b_6) も比較して増やした．**表2**にその結果と生成した NJ の数を示す．生成した NJ の数はおよそ 25×1.3^n である．最適化では解空間をすべて調べ上げたのと同じことになるが，実際に訪れたノードの数は高々反復回数 (2.2 節のアルゴリズムにおいて前の手順に戻る回数) であり，アルゴリズムの性質から反復回数は生成した NJ の数の2倍には

表 2 ロット数を増やした実験結果
Table 2 Experimental results for larger numbers of lots.

ロット数	4	5	6	7	8	9	10
無駄面積(%)	1.87	1.94	1.78	1.74	1.85	1.76	1.72
NJ の数	8	34	30	42	139	204	242

11	12	13	14	15	16	17	18	19
1.74	1.73	1.73	1.73	1.72	1.69	1.69	1.70	1.69
380	395	692	979	1,171	1,387	2,324	3,233	4,170

20	21	22	23	24	25	26	27
1.69	1.66	1.69	1.67	1.69	1.69	1.68	1.69
4,397	4,874	8,592	10,962	14,772	20,006	20,476	29,237

ほ等しいため，(実際に訪れたノードの数)/(解空間の大きさ) は 50×70^{-n} 以下となる．(解空間の大きさ)/(生成された NJ の数) を求めると，10^7 ($n=4$ の時) から 10^{48} ($n=27$ の時) まで単調に増大している．これらのデータは，NJ が解空間を効率的に絞り込んでいることを示していると考えられる．

図 4 には，n を増やした時の実行時間を他の手法の実行時間と共に示す．使用した計算機は SPARCstation (15.8 MIPS) である．本手法の実行時間はおよそ 0.02×1.5^n (秒)，元々の問題では 1 ロット 50 枚だから，1,000 枚程度の板の厳密に最適な切り方を 1 分程度で求めることが可能なことになる．オーダには指数の壁があるが，十分実用になる手法と言える．最適解が求まるまでの時間 (それ以降は最適解以上の解がないことを検証していることになる) は，全実行時間の 1/4〜1/6 となっている．例えば，$n=27$ の場合では全実行時間 1,958 秒に対して最適解が求まるまでに掛かる時間は 339 秒である．ちなみに，途中解は 6〜12 個を生成している．

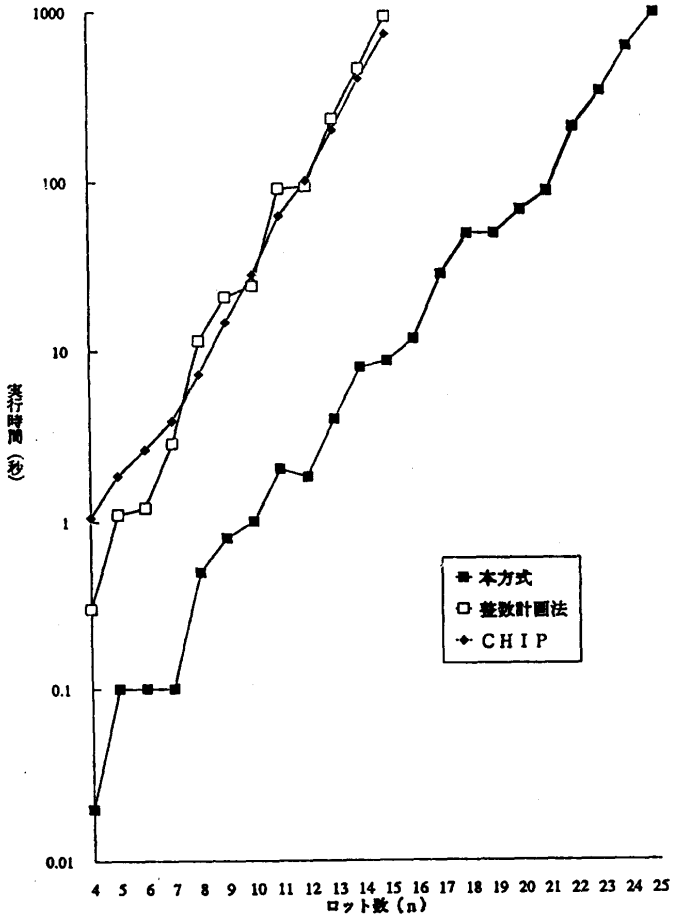


図 4 他の手法との実行時間の比較
Fig. 4 Comparison with other approaches in execution time.

図 4 の整数計画法は，C言語で記述されたプログラムによる実測値である (計算機は SPARCstation)．この問題は，不等式系および評価関数が対称式で変数値も変数間で共通な対称的な問題であるため，1 章で示した定式化より少ない 92 個の変数 x_j (各変数の取りうる値は 0 から n までの整数) で定式化できる．図 4 の CHIP は，ICL 社による CHIP の商用処理系を使って文献 3) に示されているプログラム (dichotomic search を組み込んだ改良版) を実行した実測値である．計算機は SUN 4/110 を使用したが，SPARCstation との性能差で換算した値をプロットしている．これらの手法と比べて 10 倍以上高速に最適化できることが確認された．

1 章でも触れたように，CHIP における制約の標準形は変数値の領域 (有限集合) である．等式/不等式に注目し，ある変数の変数値の中で，他の変数がその領域のいかなる値を取ろうとその式が成立しなくなるようなものを削除することにより，制約が算出される．制約が更新される (変数値の領域が狭くなる) と，他の式に伝播される．この方式では，変数値が削除されれば解空間の大きな部分が削れることが期待できる反面，「他の変数がいかなる値を取ろうと」という条件が厳しいために，なかなか制約が更新されないこともある．本方式では，ひとつの変数がいかなる値を取ろうと制約違反となって

しまうような条件 (NJ) を算出し，これをその変数以外の変数に向かって伝播する．制約違反が検出されてそれがひとつの変数だけで解消できない限り，その時点の変数値の組合せを禁止する新しい NJ を生成することが特徴である．

1章で述べた知的バックトラックおよび ATMS については，切断問題や付録2に示したナップサック問題に対しては，次の理由で，その探索空間が通常のバックトラック (chronological backtrack) と変わらなくなる．これらの問題は，不等式系も評価関数もすべての変数を含んでいるため，制約違反(矛盾)が検出された時，その時点の変数値の組合せ全体が nogood となり，その一部を nogood として特定することはできない．hyperresolution[10] によって既存の nogood から要素数の少ない nogood を生成したとしても（付録2図5のような探索木において，あるノードの各子ノードを禁止する nogood からそのノードを禁止する nogood を生成したことになる），生成された nogood の組合せを含む可能性についてはすべてチェック済みである．したがって，未探査の解空間を刈り込むことができず，探索空間は通常のバックトラックと変わらなくなってしまう．本方式では，制約違反が検出されると変数値の組合せでなく制約違反の十分条件を生成し利用することから，未探査の解空間を削ることが可能で，探索空間が小さくなる．

整数計画法の定式化における各変数の整数条件を外した緩和問題 ($n=4$) にシンプレックス法を適用したところ，$x_1=2.1333$, $x_5=0.7778$, $x_7=0.3333$, $x_{13}=0.7556$ または $x_1=2.5238$, $x_5=0.5714$, $x_7=0.3333$, $x_{10}=0.1071$, $x_{19}=0.4643$ となり（精度パラメータの値により解が異なる），最適解 ($x_1=2$, $x_2=1$, $x_{15}=1$) を求めることはできなかった．

上でも触れたように，

最適化＝最適解の探索
　　＋最適解以上の解がないことの検証　　(5)

と書ける．右辺の第2項は，最適値より厳しい制約値 b_0 に対して解空間を探索し制約充足に失敗する処理である．一方，初めから制約値 b_0 を最適値より厳しい値に設定して制約充足を行った場合は，

制約充足＝制約値を満たす解がないことの検証　　(6)

であり，(5)式の右辺第2項に対応する．しかし，実験の結果，このような制約充足の方が最適化より時間が掛かる（最大3.7倍）ケースもあることがわかった．例えば，表3には，$n=8$ の場合における最適化と制約充足のデータを示す．この最適化では，b_0 を最終的に 451,199 として制約充足に失敗し，最適解を451,200 と出力する．この表は，初めから b_0 を 451,199 や 451,000 として制約充足を行った方が最適化より時間が掛かることを示している．(5)式の右辺第2項の処理と(6)式の処理の違いは，最適解が得られるまでに生成された NJ により解空間が刈り込めるかどうかである．したがって，これらの NJ が(5)式の右辺第2項の処理を加速していると考えられる．これは，最適化において b_0 が緩やかな時に生成された NJ が有効に機能している傍証である．

表 3　制約充足と最適化
Table 3　Constraint satisfaction versus optimization.

処理	b_0*	無駄面積	NJ の数	時間(秒)
最適化	10,000,000	451,200	1,223	258
制約充足	451,199	失敗	1,318	265
制約充足	451,000	失敗	2,081	536

* 無駄になる面積に関する制約値．最適化の場合はその初期値．

4. おわりに

本論文では，制約違反の十分条件を論理式で表現した制約違反条件 (NJ) を導入し，これを用いて組合せ的な制約充足/最適化を行う手法を提案した．実験評価により，NJ が解空間の絞り込みに有効であること，他の手法と比べて10倍以上高速に最適化できること，十分実用になる手法であることを確認した．さらに規模の大きな問題に対しても，この手法が近似解法を開発するベースになりうると考えられる．

本手法は，緩和問題の解の代わりにボトムアップに作った失敗情報 (NJ) により限定操作を行う列挙法と見ることもできる．ただし，対象となる探索木は，整数計画問題として定式化した場合より小さい．

整数計画問題や制約論理型言語による定式化に対して本手法が持つひとつの利点は，初めに変数値をすべて求めておく必要はなく，必要に応じて変数値を列挙してもよいことである．したがって，状況によって不要な変数値を求める手間を省くことも可能である．もうひとつの利点は，蓄積された NJ を再利用することにより制約値を変更した制約充足/最適化が加速されることである．制約値を変えながら繰り返し制約充足/最適化を行う場合に大きな効果を期待できる．

今後の方向としては，本論文で扱った定式化が可能

な他の様々な問題に対して本手法を適用し評価を積み上げること，1つの問題についてさらに細部を詰め実用システムを目指すこと，がある．また，並列化も今後の課題である．本手法の処理のほとんどはNJの検査であり，しかも連続してチェックするNJの系列は平均して長い．したがって，例えば，複数のプロセサでNJを分け持つことによる並列処理も可能であろう．

また，本手法を積和形の論理式による制約プログラミング言語の処理系と捉えることもできる．リテラルは不等式で，その否定は不等号を逆転することで吸収する．実行中に節にあたるNJが追加されていく．このような言語としての検討も興味深い．

謝辞 本研究のきっかけは第五世代コンピュータ・プロジェクトの再委託研究であった．ICOT淵一博所長，古川康一次長，生駒憲治研究部長代理（現在NTT データ通信株式会社），新田克己第七研究室長に感謝いたします．また，多くの有益なコメントをいただいた査読者の方に感謝いたします．

参考文献

1) 溝口文雄ほか(編)：制約論理プログラミング，共立出版 (1989).
2) 茨木俊秀：アルゴリズムとデータ構造，昭晃堂 (1989).
3) Dincbas, M., Simonis, H. and Hentenryck, P. V.: Solving a Cutting-Stock Problem in Constraint Logic Programming, *Proc. of the Fifth International Conference and Symposium on Logic Programming*, pp. 42-58 (1988).
4) 丸山文宏，角田多苗子，松永裕介，箕田依子，澤田秀穂，川戸信明：評価・再設計機構を備えた論理設計支援システム，電子情報通信学会論文誌，Vol. J72-A, No. 8, pp. 1172-1180 (1989).
5) Kageyama, N., Miura, C. and Shimizu, T.: Logic Optimization Algorithm by Linear Programming Approach, *Proc. of the 27th Design Automation Conference*, pp. 345-348 (1990).
6) 今野 浩：整数計画法，産業図書 (1981).
7) 茨木俊秀：組合せ最適化—分枝限定法を中心として，産業図書 (1983).
8) Winston, P. H.: *Artificial Intelligence* (2nd ed.), Addison-Wesley (1984).
9) de Kleer, J.: An Assumption-Based TMS, *Artif. Intell.*, Vol. 28, No. 2, pp. 127-162 (1986).
10) de Kleer, J.: Extending the ATMS, *Artif. Intell.*, Vol. 28, No. 2, pp. 163-196 (1986).
11) 刀根 薫：数理計画，朝倉書店 (1978).

付録1

ここでは，非線形の例として，3章で述べた切断問題について，そのロットを構成する枚数を可変にした問題の定式化を示す．不等式系および評価関数が2次となるが，切り方とその枚数の組を変数値とすることにより，2.2節のアルゴリズムがそのまま適用できる．

$$d_7(x_1)d_1(x_1)+\cdots+d_7(x_n)d_1(x_n)\geqq b_1$$
$$d_7(x_1)d_2(x_1)+\cdots+d_7(x_n)d_2(x_n)\geqq b_2$$
$$d_7(x_1)d_3(x_1)+\cdots+d_7(x_n)d_3(x_n)\geqq b_3$$
$$d_7(x_1)d_4(x_1)+\cdots+d_7(x_n)d_4(x_n)\geqq b_4$$
$$d_7(x_1)d_5(x_1)+\cdots+d_7(x_n)d_5(x_n)\geqq b_5$$
$$d_7(x_1)d_6(x_1)+\cdots+d_7(x_n)d_6(x_n)\geqq b_6$$
$$d_7(x_1)+\cdots+d_7(x_n)\geqq b_7$$
$$d_7(x_1)+\cdots+d_7(x_n)\leqq b_7$$

次の評価関数を最小化する．

$$d_7(x_1)d_0(x_1)+\cdots+d_7(x_n)d_0(x_n)$$

ただし，b_1〜b_6は各々の棚の必要数，b_7は板の枚数，dは属性値で，d_0が無駄の面積，d_1〜d_6がそれぞれの棚の取れる枚数，d_7がロットの枚数である．

付録2

ここでは，表4に示す簡単なナップサック問題[11]を例に取り，実行トレースにより，本最適化手法についてもう一度説明する．表中で，x_1, x_2, x_3, x_4はそれぞれ品物に対応する変数であり，容量の制限（13とする）を満たして期待される価値の総和を最大にしたい．各変数の値は，ナップサックに入れていく(1)，入れていかない(0)の2つである．i番目の品物の大きさに対応する属性値を$d_1(x_i)$，期待される価値に対応する属性値を$d_0(x_i)$と書く．入れていかない時のその品物の大きさと期待される価値は0と考える．b_1は13であり，評価関数に対応する制約値(b_0)の初期値は0とする．変数の選択法はx_1からの固定順．

①2つの制約違反不等式を作る．これが初期NJ.

$$d_1(x_1)+d_1(x_2)+d_1(x_3)+d_1(x_4)>b_1 \quad (a)$$
$$d_0(x_1)+d_0(x_2)+d_0(x_3)+d_0(x_4)<b_0 \quad (b)$$

②まず，すべての品物を入れていくとする．

$x_1=1$, $x_2=1$, $x_3=1$, $x_4=1$.

表4 ナップサック問題
Table 4 A knapsack problem.

品物	x_1	x_2	x_3	x_4
大きさ	6	8	5	2
価値	1	3	7	3

(a)が成立.

③ x_1 についての変数値変更に失敗して，(a)の $d_1(x_1)$ にそれぞれ6と0を代入した条件の論理積(c)を生成.

$$0+d_1(x_2)+d_1(x_3)+d_1(x_4)>b_1 \qquad (c)$$

x_1=(不定), x_2=1, x_3=1, x_4=1.

④ x_2 についての変数値変更に成功.

x_1=(不定), x_2=0, x_3=1, x_4=1.

⑤ x_1 についての変数値割当に成功．解は，11.

x_1=1, x_2=0, x_3=1, x_4=1.

⑥ b_0 を12に更新して再評価すると(b)が成立するが，x_1 についての変数値変更に失敗して，NJ(b)の $d_0(x_1)$ にそれぞれ1と0を代入した条件の論理積(d)を生成.

$$1+d_0(x_2)+d_0(x_3)+d_0(x_4)<b_0 \qquad (d)$$

x_1=(不定), x_2=0, x_3=1, x_4=1.

⑦ x_2 についての変数値変更にも失敗して，NJ(d)の $d_0(x_2)$ に0を代入した条件と NJ(c)の $d_1(x_2)$ に8を代入した条件の論理積（e）を生成.

$$1+d_0(x_3)+d_0(x_4)<b_0$$
$$\wedge 8+d_1(x_3)+d_1(x_4)>b_1 \qquad (e)$$

x_1=(不定), x_2=(不定), x_3=1, x_4=1.

⑧ x_3 についての変数値変更に成功.

x_1=(不定), x_2=(不定), x_3=0, x_4=1.

⑨ x_2 についての変数値割当に失敗して，NJ(d)の $d_0(x_2)$ にそれぞれ3と0を代入した条件の論理積(f)を生成.

$$4+d_0(x_3)+d_0(x_4)<b_0 \qquad (f)$$

⑩ x_3 についての変数値変更に失敗して，NJ(e)の $d_0(x_3)$ に7, $d_1(x_3)$ に5を代入した条件と NJ(f)の $d_0(x_3)$ に0を代入した条件の論理積(g)を生成.

$$8+d_0(x_4)<b_0 \wedge 13+d_1(x_4)>b_1 \qquad (g)$$

x_1=(不定), x_2=(不定), x_3=(不定), x_4=1.

⑪ x_4 についての変数値変更に成功.

x_1=(不定), x_2=(不定), x_3=(不定), x_4=0.

⑫ x_3 についての変数値割当に失敗して，NJ(f)の $d_0(x_3)$ にそれぞれ7と0を代入した条件の論理積(h)を生成.

$$11+d_0(x_4)<b_0 \qquad (h)$$

⑬ x_4 についての変数値変更に失敗して，NJ(g)の $d_0(x_4)$ に3, $d_1(x_4)$ に2を代入した条件と NJ(h)の $d_0(x_4)$ に0を代入した条件の論理積(i)を生成.

$$11<b_0 \wedge 15>b_1 \qquad (i)$$

変数を含まない NJ(i) が生成されたためアルゴリズムは終了する．解(11)が1つ得られ，それ以上の解が見つからなかったため，それ(11)が最適解.

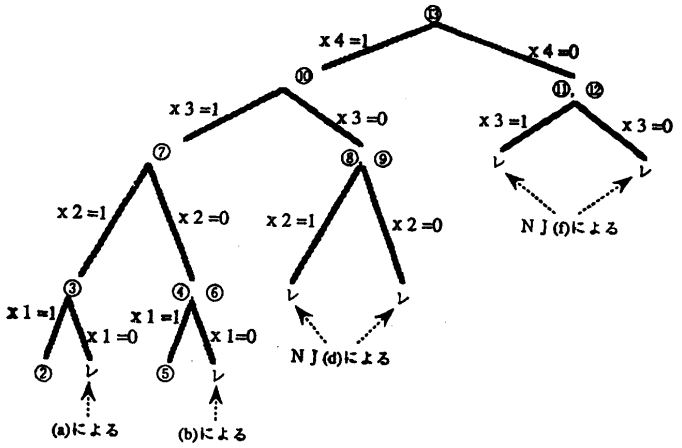


図5 ナップサック問題における探索
Fig. 5 Search for the knapsack problem.

x_1=1, x_2=0, x_3=1, x_4=1. ここで(i)は，容量が15未満に制限されていると期待される価値の総和は11より大きくはできない，ことを意味する.

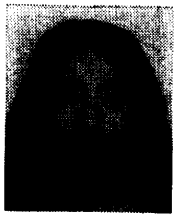
以上の経過を図で示すと図5のとおりとなる．図中，レは，NJ によりそれ以下の探索を省略できたノードを示す．NJ(d) は，もともと ATMS における nogood $\{x_2=0, x_3=1, x_4=1\}$ として生成されたが，nogood $\{x_2=1, x_3=0, x_4=1\}$ や nogood $\{x_2=0, x_3=0, x_4=1\}$ としても機能しており，NJ(f) は，もともと nogood $\{x_3=0, x_4=1\}$ として生成されたが，nogood $\{x_3=1, x_4=0\}$ や nogood $\{x_3=0, x_4=0\}$ としても機能している．したがって，これらの NJ により探索が省略できた部分が ATMS による探索空間との差となる.

CHIP と比較すると，CHIP では，⑤までの間はどの変数の変数値も削除できない．したがって，制約の伝播も起きない．これに対して，この実行トレースでは，③で NJ(c) が生成されており，これ以降の処理に利用されている．また，CHIP では容量をどれほど増やせばさらに解(11)が改善されるかの情報は得られないが，上では NJ(i) がそれを与えている.

(平成3年2月18日受付)
(平成4年3月12日採録)

丸山　文宏（正会員）

1955 年生．1978 年東京大学工学部計数工学科卒業．同年(株)富士通研究所入社．1981～82 年スタンフォード大学計算機科学科客員研究員．工学博士．現在，(株)富士通研究所知識処理研究部．筑波大学非常勤講師．CAD，人工知能の研究に従事．1980 年本学会創立 20 周年記念論文賞．1988 年元岡賞．電子情報通信学会，人工知能学会，日本ソフトウェア科学会，IEEE 各会員．



箕田　依子（正会員）

1962 年生．1985 年横浜国立大学工学部金属工学科卒業．同年，富士通(株)入社．知識処理の研究に従事．

澤田　秀穂（正会員）

1962 年生．1986 年東京農工大学工学部数理情報工学科卒業．同年，富士通(株)入社．知識処理の研究に従事．

滝沢　ユカ（正会員）

1965 年生．1989 年津田塾大学学芸学部数学科卒業．同年富士通(株)入社．

川戸　信明（正会員）

昭和 23 年生．昭和 46 年東京大学工学部電子工学科卒業．昭和 51 年同大学院工学系研究科電子工学専門課程博士課程修了．工学博士．同年(株)富士通研究所入社．以来，論理装置の CAD の研究開発に従事．AI 技術や並列処理技術の CAD への応用に特に興味をもつ．電子情報通信学会，日本ソフトウェア科学会，人工知能学会，IEEE 各会員．