Vol. 35 No. 12 情報処理学会論文誌 Dec. 1994

最小矛盾の概念を用いた混合 0-1 整数計画問題の近似解法

原 裕貴†

混合 0-1 整数計画問題は，長くオペレーションズリサーチの分野で研究されてきた重要な問題であるが，一般に NP-complete であり厳密に解くことは難しい．本論文では，混合 0-1 整数計画問題に対する新しい近似解法を提案する．本手法では，0-1 変数に対して近傍探索を行い，連続変数に対しては，シンプレックス法を用いて最適化を行う．近傍探索の特長は最小矛盾の概念を用いる点である．最小矛盾は，制約違反を引き起こすような最小の仮説の集合である．最小矛盾を用いることによって，近傍探索を効率良く実行でき，また局所最適の問題も生じない．本手法を典型的な混合 0-1 整数計画問題である多品種生産計画問題に適用し，分枝限定法や従来のヒューリスティクス手法と比較し，本手法の有効性を示した．

An Approximate Method for Mixed 0-1 Integer Programming Problem Using Minimal Conflict

HIROTAKA HARA†

We propose an approximate method for the mixed 0-1 integer programming problem. We combine a local search method on 0-1 variables and the simplex method on continuous variables. Our idea is to use the minimal conflict. The minimal conflict is a minimal set of assumptions that causes a constraint violation. The use of the minimal conflict makes the local search efficient and avoids the local optimality. We applied this method to multi-item production scheduling problems and demonstrated its effectiveness compared with the branch and bound method and several heuristic methods.

1. はじめに

組合せ最適化は，巡回セールスマン問題，ナップザック問題，スケジューリング問題，最適配置問題等幅広い応用を含み，古くからオペレーションズリサーチの分野で研究されてきた．一方，AI の分野でも組合せ的な問題は古くから扱われており，TMS[3] や ATMS[5] もそのような手法の一つと考えることができる．近年では，このような問題が CSP[6] として定式化され，各種の手法が研究されている[7),10]．しかし，AI 分野では制約を満たす解を求めることが主な目的であり，最適化を陽に扱うことはなかった．

我々は，組合せ最適化に対して最小矛盾の概念を用いた近似解法の研究を行っており，すでに典型的な組合せ最適化であるジョブショップスケジューリング問題および全 0-1 整数計画問題に対して適用した結果を報告をしている[4),12]．本稿では，我々の方式を混合 0-1 整数計画問題に適用した結果について報告する．

† (株)富士通研究所知識処理研究部
Intelligent Systems Laboratory, Fujitsu Laboratories Ltd.

混合 0-1 整数計画問題は以下のように定式化される[8]．

$$\begin{aligned} z=\max\quad & cx+hy \\ \text{subject to}\quad & Ax+Gy\leq b \\ & x\in\{0,1\}^n \\ & 0\leq y \end{aligned} \qquad (1)$$

ここで，A は $m\times n$ 行列，G は $m\times l$ 行列，c は $1\times n$ 行列，h は $1\times l$ 行列，b は $m\times 1$ 行列，x は $n\times 1$ 行列，y は $l\times 1$ 行列である．

以下に混合 0-1 整数計画問題の簡単な例を示す．

$$\begin{aligned} z=\max\quad & 2x_1+3x_2+2y_1+y_2 \\ \text{subject to}\quad & 2x_1-3x_2+y_1-4y_2\leq 3 \\ & 2x_1+4x_2+3y_1+y_2\leq 8 \\ & x_1,\ x_2\in\{0,1\} \\ & 0\leq y_1, y_2 \end{aligned} \qquad (2)$$

混合 0-1 整数計画問題は，生産計画問題，配置問題，スケジューリング問題など多くの応用があり，重要な問題である．混合 0-1 整数計画問題に対する厳密な解法としては分枝限定法，切除平面法などがあるが，一般に混合 0-1 整数計画問題は NP-complete であり，問題の規模が大きくなると厳密に解くことは極

めて難しくなる．このため，個別の問題ごとに各種の近似手法が開発されている．本論文では混合0-1整数計画問題一般に適用可能な汎用的な近似解法を提案する．

2. 最小矛盾を用いた最適化方式

我々の方式は，近傍探索法の一種である．すなわち，すべての0-1変数に0か1を割り当てて初期解を生成し，ある0-1変数の値を0から1（または1から0）へと変更していくことを繰り返していく．すべての0-1変数に値を割り当てると，残りの変数は連続変数だけとなる．この問題は線形計画問題の形をしており，Simplex法を利用することによって効率的に解くことができる．

我々のアイデアは最小矛盾（この用語は文献7）で用いられているが，我々の定義とは異なる）の概念を用いることである．

以下では，いくつかの用語を定義する．

仮定（assumption）は，0-1変数に対する0-1値の代入，すなわち $x_i=v_i$ $(v_i\in\{0,1\})$ である．

割当（assignment）は，仮定の集合 $\{x_i=v_i : i\in N^*\subset\{0,1,\cdots,n\},\ v_i\in\{0,1\}\}$ である．

解（solution）は，すべての0-1変数に対する仮定の集合 $\{x_i=v_i : i\in\{0,1,\cdots,n\},\ v_i\in\{0,1\}\}$ であり，解 S に対して $x^s=(v_1,v_2,\cdots,v_n)$ を**解ベクトル**と呼ぶ．

解 S に対して，LP(x^s) を以下のように定義する．

$$z_{LP}(x^s)=\max\ hy$$
$$\text{subject to}\quad Gy\leq b-Ax^s$$
$$0\leq y \qquad (3)$$

LP(x^s) は整数変数を含まない線形計画問題であり，良く知られたSimplex法を用いることによって効率良く解くことができる．LP(x^s) が解をもつとき，解 S を**制約充足解**と呼び，そうでないとき**制約非充足解**と呼ぶ．

問題(1)の最適解を (x_0,y_0)，その時の目的関数値を z_0 とすると，LP(x_0) の最適解が y_0 で，$z_0=cx_0+z_{LP}(x_0)=cx_0+hy_0$ であることは明らかである．

不等式 $ax\leq(<)b$ に対して，$ax^s>(\geq)b$ であるとき，この不等式を解 S に対する**違反制約**と呼ぶ．制約違反を引き起こすような最小の割当 $(\subset S)$ を**最小矛盾**(minimal conflict) と呼ぶ．

例1

解 $\{x_1=1,\ x_2=0,\ x_3=1,\ x_4=1\}$ に対し不等式 $5x_1-2x_2-x_3+2x_4\leq 3$ は違反制約である．このとき，割当 $\{x_1=1,\ x_2=0\}$ は最小矛盾である．なぜなら x_3, x_4 をどのように割り当てても不等式を満たすことは出来ない．$\{x_1=1\}$ は最小矛盾ではない．なぜなら，例えば $\{x_2=1,\ x_3=1,\ x_4=0\}$ を割り当てることによって，この不等式を満たすことができる．$\{x_1=1,\ x_4=1\}$ もまた別の最小矛盾である．

LP(x^s) が解をもつとき，双対問題の最適解を u とすると，双対定理から，$z_{LP}(x^s)=u(b-Ax^s)$ と表すことができる[8]．元の問題に対して，それまでに求められた最大の目的関数値を *BestValue* とすると，不等式 $cx+u(b-Ax)>BestValue$ は解 S に対する違反制約になる．

LP(x^s) が解をもたないとき，$v(b-Ax^s)<0$ を満たすような双対レイ (dual ray) v が存在する[8]．このとき，不等式 $v(b-Ax)\geq 0$ は解 S に対する違反制約になる．

本手法のアルゴリズムを以下に示す．

アルゴリズム A

(1) 初期解を生成し，S とする．
MinimalConflictSet$=\{\}$.
BestValue$=-\infty$.

(2) 停止条件が満足されたら，*BestSolution* と *BestValue* を出力して停止する．

(3) LP(x^s) を解く．

(a) LP(x^s) が解をもたないならば，$v(b-Ax^s)<0$ を満たすような双対レイを求め，v とする．

(b) LP(x^s) が解をもつならば，その双対解を u とする．

(4) LP(x^s) が解をもち，*BestValue*$<cx^s+z_{LP}(x^s)$ ならば，*BestSolution*$=S$, *BestValue*$=cx^s+z_{LP}(x^s)$ とする．

(5) 違反制約を発見する．

(a) LP(x^s) が解をもたない場合
$v(b-Ax)\geq 0$ を違反制約とする．

(b) LP(x^s) が解をもつ場合
$cx+u(b-Ax)>BestValue$ を違反制約とする．

(6) (5)で求めた違反制約の最小矛盾を求め，*MC* とする．

(7) *MinimalConflictSet* に *MC* を追加する．

(8) *MC* の中から一つの仮説 $x_r=v_r$ を選ぶ．この仮説の値を反転して新しい解 S' を生成する．すなわち，$S'=\{x_i=v_i' : i\in\{1,2,\cdots,n\},\ v_r'=\tilde{}v_r,$

$v_i'=v_i$ $(i\neq r)\}$. ただし, $v=1$ ならば $\tilde{}v=0$, $v=0$ ならば $\tilde{}v=1$.

(9) S' が *MinimalConflictSet* に含まれる最小矛盾の一つでも含んでいたら, (5)に戻り別の仮説を選ぶ.

(10) S' を新しい解として S と置き換え, (2)に戻る.

アルゴリズムの特徴

(1) 最小矛盾の意味

解が制約非充足の場合の最小矛盾は, 解が制約非充足のための十分条件を示している. したがって, 最小矛盾に含まれる仮説の少なくとも一つを変更することが, 解が制約充足するための必要条件となる. 解が制約充足の場合の最小矛盾は, 解の目的関数値が *BestValue* より悪いための十分条件になっている. したがって, 最小矛盾に含まれる仮説の少なくとも一つを変更することが, 解の目的関数を *BestValue* 以上に改善するための必要条件となる.

(2) 山登り法との比較

近傍探索として良く用いられる山登り法の場合, 制約充足解のみを対象として, 目的関数が改善される方向に変更を繰り返す. このため, 制約が強く制約充足解が少なくなると, 局所最適解に陥って停止する問題点が生じる. 本方式は, 目的関数が悪くなる方向への変更も受理されるので, 局所最適解で停止することはない. また, 制約非充足解も探索空間中にあるため, 制約が強い問題でも効率良く動作する.

(3) ループの回避

本方式では, 生成された解に対して必ず最小矛盾が生成, 保存され, その後新しく生成された解に対してはこの最小矛盾とのチェックが行われるため, 再び同じ解が生成されることはない.

(4) 探索空間の枝刈り

最小矛盾は, 生成された解以外にも, その最小矛盾を含む解が生成されることを防ぐ. その意味において, 無駄な探索空間を枝刈りする効果がある.

二重反転

上のアルゴリズムのステップ5において生成されるすべての仮説が, ステップ6において受理されないことがある. この場合は, 二つの仮説を同時に反転することによって新たな解を生成する. このようなことは非常に稀なので, 全体の効率にはさほど影響しない.

ヒューリスティクスの利用

上のアルゴリズム中, 二つの部分でヒューリスティクスを利用することができる. 一つは初期解の生成であり, もう一つはステップ8で最小矛盾の中から変更する仮説を選ぶ部分である.

我々は, 元の0-1整数計画問題から整数制約を除いたLP緩和問題の解を利用している. LP緩和問題は, シンプレックス法を用いて効率的に解くことができる. LP緩和問題は以下のように定義される.

$$\begin{aligned} z=\max\quad & cx+hy \\ \text{subject to}\quad & Ax+Gy\leq b \\ & 0\leq x\leq 1 \\ & 0\leq y \end{aligned} \qquad (4)$$

初期解としては, LP緩和解を丸めたものを使う(この解は制約充足解とは限らないことに注意). ステップ8では, LP緩和解に最も近くなるような反転から優先的に試す. 例えば, 最小矛盾が $\{x_1=0, x_2=1\}$ で, この変数に対するLP緩和解が $\{x_{lp1}=0.7, x_{lp2}=0.2\}$ であるとすると, 反転値 $x_2=0$ と $x_{lp2}=0.2$ との差は, 反転値 $x_1=1$ と $x_{lp1}=0.7$ との差より小さいので, x_2 の反転を先に試す. この評価値が同じときはランダムに選択する.

次に, 本アルゴリズムを山登り法と組み合わせたアルゴリズムについて述べる. 変更点は以下のようである. ステップ8において, 目的関数が悪くならない変更のみを受理する. 予め定めておいた定数 *mcmax* の仮説を生成してもそのような仮説が見つからない場合は, それまで生成された仮説の中でもっとも目的関数値の良いもの(同じ値の場合はランダムに選ぶ)を次の解とする. アルゴリズムの変更点を以下に示す. ただし, obj(S) は, S が制約充足の場合は cx^s, 非充足の場合は $-\infty$ を返す関数である.

アルゴリズムB(アルゴリズムAとの変更点のみ)

(8) $i=0$, *candidateset*$=\{\}$ として, *MC* 中のすべての仮説について以下を行う.

(8.1) *MC* の中から一つの仮説 $x_r=v_r$ を選ぶ. ここの仮説の値を反転して新しい解 S' を生成する. すなわち, $S'=\{x_i=v_i' : i\in\{1,2,\cdots,n\}, v_r'=\tilde{}v_r, v_i'=v_i (i\neq r)\}$. ただし, $v=1$ ならば $\tilde{}v=0$, $v=0$ ならば $\tilde{}v=1$.

(8.2) S' が *MinimalConflictSet* に含まれる違反制約の一つでも含んでいたら, (8.1) に戻り別の仮説を選ぶ.

(8.3) obj(S')$\geq$obj(S) ならば, (10)へ.

(8.4) $i=i+1$ とする. $i\geq$*mcmax* ならば, (9)へ. そうでなければ, S' を *candidateset* に追加し, (8.1) に戻り別の仮説を選ぶ.

(9) *candidateset* の要素中で obj(S') が最大のものを S' として(10)へ.

3. インプリメント

3.1 最小矛盾発見アルゴリズム

ステップ6における最小矛盾発見アルゴリズムは,変数の数 n に対して, $O(n\log n)$ で構成できる.

違反制約は,

$$c_1x_1+c_2x_2+\cdots+c_nx_n<(\leq)rhs$$

と記述することができる. 不等号の向きが逆のときは,両辺に -1 をかけて反転する.

まず,係数がすべて正であると仮定する. このときは, $x_i=1$ であるような項を,係数 c_i の大きい順に,不等式を満たさなくなるまで選択すれば良い. 係数が同じ値の時はランダムに選択する. c_i のソートにかかる手間が $O(n\log n)$ であり,選択にかかる手間は $O(n)$ である.

例2

解が $\{x_1=1,\ x_2=1,\ x_3=0,\ x_4=1,\ x_5=1\}$ で,違反制約が $2x_1+6x_2+5x_3+x_4+3x_5<11$ であるとする. ソートすると, $6x_2+3x_5+2x_1+x_4<11$ となる ($x_3=0$ なので, x_3 の項は除く). $6+3<11\leq 6+3+2$ なので,最小矛盾は, $\{x_2=1,\ x_5=1,\ x_1=1\}$ である.

係数が負の場合は,以下の操作によって正にかえる. $c_1x_1+\cdots+c_ix_i+\cdots+c_nx_n<(\leq)rhs$ において, $c_i<0$ とする. このとき, $x_i'=1-x_i$ とおくと, $c_1x_1+\cdots-c_ix_i'+\cdots+c_nx_n<(\leq)rhs-c_i$ となる.

この操作に要する手間は,最悪でも $O(n)$ であり,結局,全体の手間は, $O(n\log n)$ である.

3.2 最小矛盾のデータ構造

本手法では,多くの最小矛盾が生成され保存される. したがって,最小矛盾を小さく表現することが,メモリ効率上重要である.

このような矛盾をビットベクトルで管理する手法はATMS でも行われており[11], 我々の用いた手法も基本的には同じである.

通常の ATMS では, $x_i=1$ という仮説と $x_i=0$ という仮説は別の仮説として表現され,長さ $2n$ のビットベクトルが用いられるが,我々は最小矛盾を二つの n-ビットベクトル v_0, v_1 のペアで表現した. v_1 の i ビット目が1であるのは, $x_i=1$ を表し, v_0 の i ビット目が1(0ではないことに注意)であるのは, $x_i=0$ を表す. このビットベクトルペアを $\mathrm{mc}(v_0, v_1)$ と表現する.

例3

$n=5$ で,最小矛盾が $\{x_1=0,\ x_3=1,\ x_4=0\}$ であるとする. この最小矛盾をビット表現すると, (v_0=10010, v_1=00100) である.

$n=1000$ で,最小矛盾が 1024 個たまった場合のメモリ消費量は, 2·1000·1024 bits=256000 bytes=250 Kbytes である.

この表現を利用すると,ステップ9における最小矛盾の包含チェックをビット操作によって高速に実行することができる. すなわち,解 S のビット表現を v_s とすると, $(\sim v_s\ \&\ v_1)=0$ かつ $(v_s\ \&\ v_0)=0$ であれば, v_s は $\mathrm{mc}(v_0, v_1)$ を包含し,そうでなければ包含しない. ただし, $\sim$ と & はそれぞれビット否定,ビット積を表す.

例4

$S=\{x_1=0,\ x_2=1,\ x_3=1,\ x_4=1,\ x_5=0\}$ とする (すなわち, v_s=01110). このとき, (~01110&00100)=00000 であるが, (01110&10010)=01000 なので, S は例3の最小矛盾 $\{x_1=0,\ x_3=1,\ x_4=0\}$ を包含しない.

3.3 最小矛盾集合の管理

ステップ9においては,生成されたすべての最小矛盾について包含チェックを行う必要がある. 個々の包含チェックは前記の手法によって高速に行うことができるが,すべての最小矛盾とチェックを行うのは効率が良くない. 我々は,次の性質を利用してチェックする最小矛盾の数を減らしている.

ベクトル v 中の1の個数を $|v|_1$, 0の個数を $|v|_0$ と書くとする. 解 S, 最小矛盾 $\mathrm{mc}(v_0, v_1)$ に対して, $|v_s|_1<|v_1|_1$ であれば, S は mc を包含することはできない. 同様に, $|v_s|_0<|v_0|_1$ であっても, S は mc を包含することはできない.

我々は,最小矛盾集合を $|v_0|_1$ と $|v_1|_1$ の値によって分類,管理している. 上の性質によって,解 S に対しては, $|v_1|_1\leq|v_s|_1$ かつ $|v_0|_1\leq|v_s|_0$ を満たす最小矛盾 $\mathrm{mc}(v0, v1)$ だけをチェックを行えば良い ($|v_s|_1+|v_s|_0=n$ であることに注意せよ).

4. 実験結果

我々の手法を評価するために,典型的な混合0-1整数計画問題である多品種生産計画問題を対象として他の手法との比較を行った. この問題はp品種の生産物のt期間の生産計画を立てる問題で,各期間に各品種を生産するかどうかが0-1変数に対応する. 各期間の

各品種の生産量と在庫量が連続変数となる．各期間ごとに資源に関する制約があり，目的関数は，セットアップコスト，在庫コストの和の最小化である．この問題の正確な定義については，付録Aを参照のこと．

今回の実験では，整数計画問題に対する一般的な手法である分枝限定法およびこの問題専用にORの分野で開発されたいくつかのヒューリスティックとの比較を行った．なお以下の測定ではいずれもアルゴリズムBを利用している．また，Simplex 法の実行には商用のパッケージ CPLEX[2] を利用した．

分枝限定法

分枝限定法は，整数計画問題に対する一般的で正当的な手法である．分枝限定法は，LP 緩和解からスタートして，ある変数の値を0か1に固定する分枝操作と，最適解を生成する可能性がなくなったノードを終端する限定操作からなる．分枝限定法は，すべての探索木の探索を終了すれば必ず最適解が求められ，その最適性が保証されるが，問題のサイズが大きくなると，探索木の大きさは一般に指数関数的に増大し，計算時間の点でも，メモリ効率の点でも，実際に探索を終了することは不可能となる．この場合，適当な時間で探索を打ち切って，その時点までに求まった最良解を近似解として利用することになる．今回の測定では，分枝限定法を実行するのに商用のパッケージ CPLEX[2] を利用した．

まず，8品種8期間の問題に対して測定を行った．問題は四つあり，これらの問題のデータは文献 9) から取り上げた．四つの問題は基本的には同じ問題であるが，各期間の資源の供給量が異なる．問題1が最も資源の供給が少なく，問題4が最も多い．言い換えると，問題1が最も制約が厳しく，問題4が最も制約が緩い．このデータの詳細については，付録Bを参照のこと．

また文献 9) には，それぞれの問題に対する最適値と三つのこの問題専用のヒューリスティックによって求められた解が記述されている．**表1**にこれらの結果と我々の手法の結果を示す．

表中，Lanbrecht VanderVeken, Sixon Silver, Subgradient の三つが文献 9) からの引用である．いずれも IBM 3081 で実行され非常に高速に解を求めているが，Data 4 に対する Subgradient を除いていずれも最適解を求めていない．特に，Data 1 に対しては，どの手法の解もかなり悪い．

我々のアルゴリズムは，乱数を利用する部分を含むため，各問題に対して異なった乱数系列による5回の測定を行った．表には平均のCPU時間と最良のCPU時間を示した．測定は，Sun workstation ELC (16 Mbyte memory) を用いて行った．$mcmax=20$ に設定した．実行マシンが違うので単純な比較はできないが，我々の手法は，文献 9) の3種法に比べて遅い．しかし，いずれの問題に対しても（異なる五つの乱数系列すべてにおいて），最適解を発見することができた．Data 1 に対しては，最適解 8430 を見つけるのに平均で 1760 秒（最良の場合は，約 100 秒）かかっているが，準最適な解 8490 は，平均約 150 秒で発見している．この解は文献 9) の3手法の解に比べてかなり優れている．

分枝限定法については，まず最良優先探索を行った．この探索は，大量のノードを生成しメモリを消費する傾向がある．このため，頻繁にスワップが生じ，

表 1 8品種8期間生産計画問題に対する測定結果
Table 1 Computational result on 8 products 8 periods problems.

Data	Optimal Value	Hara #			Lanbrecht $ VanderVeken		Dixon Silver $		Subgradient $		Branch&bound (best first) #		Branch&bound (depth first) #	
		Value	Ave. CPU(s)	Best CPU(s)	Value	CPU(s)	Value	CPU(s)	Value	CPU(s)	Value	CPU(s)	Value	CPU(s)
Data 1	8430	*8430	1759.68	98.88	8970	0.0023	8710	0.0068	8710	0.0068	9910	980	9870	6206
		8490	146.69	78.95										
Data 2	7910	*7910	533.73	332.41	8800	0.002	7930	0.0081	7920	0.008	9430	626	8770	6476
Data 3	7610	*7610	108.75	44.48	7970	0.002	7970	0.0072	7660	0.724	9530	631	9440	797
Data 4	7520	*7520	27.21	18.63	8000	0.0024	8000	0.0072	*7520	0.732	9540	357	9530	6104

Sun workstation ELC (16Mbyte memory)

$ IBM 3081, CMS 2.0, FORTHX

* Optimal value

表 2 50品種16期間生産計画問題に対する測定結果
Table 2 Computational result on 50 products 16 periods problems.

Data	Hara		Branch&bound (best first)		Branch&bound (depth first)	
	Ave. Value	Ave. CPU(s)	Value	CPU(s)	Value	CPU(s)
Data 1	3280194	626.2	3893975	1428	3924727	7201
	3153206	9226.4				
Data 2	3339355	316.4	4311251	409	4300830	13950
	2763184	17109.8				

Sun workstation ELC (16Mbyte memory)

応答性能を著しく低下させることがある．我々の測定では応答時間が10時間（CPU時間で500秒から1500秒）を越えたところで測定を打ち切った．表には，測定中に発見された最良の解と発見された時間を示している．いずれの問題に対しても非常に悪い解しか発見していない．

次に，深さ優先探索による分枝限定法を実行した．深さ優先探索の場合は，メモリを大量に消費することはない．CPU時間で2時間を越えたところで測定を打ち切った．この場合でも，発見した解の質は非常に悪い．

次に，50品種16期間の問題を作成し，我々の手法と分枝限定法との比較を行った．ここでも，資源供給量の異なる二つの問題について測定を行った．data 1は供給量が少なく，data 2は供給量が多い．我々の手法と分枝限定法の結果を**表2**に示す．

我々の手法では，$mcmax=200$ に設定し，CPU時間で5時間（18000秒）実行したところで打ち切った．各問題に対する上段の結果は最初に発見された制約充足解で，下段は測定中発見された最良解である．先の測定と同様に，各問題について5回実行を行い，平均の値を示している．

分枝限定法もCPU時間で5時間実行したところで打ち切り，表には発見された最良解を示している．ただし最良優先探索においては，メモリ不足のために測定半ばで実行が強制終了してしまった（Data 1では約13000秒，Data 2では約3000秒）．

我々の手法は分枝限定法に比べて，data 1では約20%，data 2では約35%良い解を発見している．

5. 関連研究（Pivot and Complement 法）

0-1整数計画法に対する汎用の近似解法としては，Pivot and Complement 法がある[1]．この手法は，二つのフェーズから構成される．フェーズ1では，LP緩和解からスタートして，ピボット操作を用いて非基底のスラックス変数を基底にすることによって，整数条件を満足する解を求める．フェーズ2では，フェーズ1で求められた整数解に対して，一度に三つの変数値の変更までを許す山登り法を適用して，解の改善を行う．フェーズ1は基本的に実数領域を対象としており，我々の手法とは異なる．フェーズ2は単純な山登り法である．Pivot and Complement 法のフェーズ1の結果を初期解として，我々の手法を適用することは可能である．

Pivot and Complement 法は，ランダムに作成された全0-1整数計画に対しては非常に良い解を高速に発見することが報告されている．しかし，(1)構造をもつ問題に対しては解の質が悪くなる，(2)混合0-1整数計画には直接は適用できない，というような問題点がある．

6. おわりに

混合0-1整数計画問題に対する新しい近似解法を提案した．本手法は問題固有のヒューリスティックによらない汎用的な手法であり，厳密に解くことが難しい大規模な混合0-1整数計画問題に対する実用的な手法であると考えている．多品種生産計画問題を対象とした今回の測定では，良い結果を得ることができたが，さらに別にタイプの問題に適用することによって，本手法の有効性を確認する予定である．

今後の研究課題としては，一般の混合整数計画問題への拡張，分枝限定法との組み合わせなどが考えられる．

謝辞 MIT留学中，良好な研究環境を提供していただき，また本論文に対しても有効な助言を与えていただいた J. F. Shapiro 教授に感謝いたします．

参考文献

1) Balas, E. and Martin, H.: Pivot and Complement—A Heuristic for 0-1 Programming, Management Science, Vol. 26, No. 1, pp. 86-96 (1980).
2) CPLEX Reference Manual, CPLEX Optimization Inc. (1990).
3) Doyle, J.: A Truth Maintenance System, *Artif. Intell.*, Vol. 12, No. 3, pp. 231-272 (1979).
4) Hara, H.: Solving the Large-scale 0-1 Integer Programming Proglem Using an Assumption-based Method, *Proceedings ECAI-92*, pp. 124-128 (1992).
5) de Kleer, J.: An Assumption-based TMS, *Artif. Intell.*, Vol. 28, No. 2, pp. 127-162 (1986).
6) Mackworth, A.K. and Frender, E.C.: The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems, *Artif. Intell.*, Vol. 25, No. 1, pp. 65-73 (1985).
7) Minton, S., Johnston, M.D., Philips, A.B. and Laird, P.: Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method, *Proceedings AAAI-90*, pp. 17-24 (1990).
8) Nemhauser, G. and Wolsey, L.: *Integer and Combinatorial Optimization*, John Wiley & Sons (1988).
9) Thizy, J.M. and Wassenhove, L.N.: Lagrangean Relaxation for the Multi-Item Capacitated Lot-Sizing Problem: A Heuristic Implementation, *IIE Trans.*, Vol. 17, No. 4, pp. 308-313 (1985).
10) Zweben, M. and Eskey, M.: Constraint Satisfaction with Delayed Evaluation, *Proceedings IJCAI-89*, pp. 875-880 (1989).
11) 奥野 博：ATMS の高速化技法とその応用，人工知能学会誌，Vol. 6, No. 1, pp. 24-37 (1991).
12) 原 裕貴，湯上伸弘，吉田裕之：仮定に基づく組み合わせ最適化手法，情報処理学会論文誌，Vol. 33, No. 5, pp. 595-601 (1992).

付録A 多品種生産計画問題

$$z=\min \sum_{i\in I}\sum_{t\in T} s_{it}\ y_{it}+h_{it}\ I_{it}$$

$$I_{it}=I_{i,t-1}+x_{it}-d_{it} \quad (\text{for all } i\in I, t\in T)$$

$$\sum_{i\in I} a_i\ x_{it}\le c_t \quad (\text{for all } t\in T)$$

$$x_{it}\le y_{it}\sum_{t\in T} d_{it} \quad (\text{for all } i\in I, t\in T)$$

$$x_{it}, I_{it}\ge 0 \quad (\text{for all } i\in I, t\in T)$$

$$I_{i0}=0 \quad (\text{for all } i\in I)$$

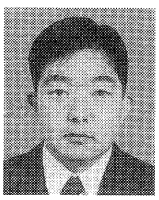
$$y_{it}=0 \text{ or } 1 \quad (\text{for all } i\in I, t\in T)$$

I：製品の集合
T：生産期間の集合
s_{it}：製品 i の期間 t におけるセットアップコスト
h_{it}：製品 i の期間 t における在庫コスト
d_{it}：製品 i の期間 t における必要量
c_t：期間 t における資源の供給量
a_i：製品 i 1単位あたり資源消費量
x_{it}：製品 i の期間 t における生産量（連続変数）
I_{it}：製品 i の期間 t における在庫量（連続変数）
y_{it}：製品 i を期間 t に生産するならば1，そうでなければ0（0-1 変数）

（平成4年9月17日受付）
（平成5年6月17日採録）

付録B 8品種8期間生産計画問題のデータ

				t							
				1	2	3	4	5	6	7	8
product i	s_{it}	h_{it}	a_i	d_{it}							
1	100	1	1	-	70	50	100	20	80	-	100
2	200	1	1	20	40	50	10	30	-	40	50
3	200	1	1	40	50	-	100	40	80	90	160
4	300	1	1	-	100	100	150	160	90	100	100
5	400	1	1	50	-	20	40	10	10	20	10
6	250	1	1	70	40	40	40	100	20	40	50
7	500	1	1	-	20	50	10	20	60	40	40
8	300	1	1	10	20	-	-	10	10	20	30
Data				c_t							
Data 1				350	350	350	400	400	400	400	500
Data 2				400	400	400	400	400	400	400	400
Data 3				500	500	500	500	500	500	500	500
Data 4				600	600	600	600	600	600	600	600



原 裕貴（正会員）

1961年生．1984年東京大学理学部情報処理学科卒業．同年(株)富士通研究所入社．1991年～92年 MIT 客員研究員．人工知能ソフトウェア．特に診断，計画などの推論システムの研究，開発に従事．情報処理学会第45回全国大会奨励賞，第7回人工知能学会全国大会優秀論文賞受賞．人工知能学会，IEEE 各会員．