制約条件に論理的ORを含む組合せ最適化問題に対するハイブリッド型最適化手法の実現

大西 秀志　田村 直之

組合せ最適化問題に対する主なアプローチに大域探索と局所探索がある．本論文では，これらを組み合わせた新しいハイブリッド型の最適化手法を提案する．提案手法は，(1) 問題の制約条件中に含まれる論理的 OR を利用して解空間を細分化し，(2) それらの小空間の間を移動しながら，(3) 大域探索法を用いて訪れた各小空間の最適解を順に求める．これにより，制約条件に OR を含むどのような問題に対しても，大域探索ソルバのみを用いて局所探索的に準最適解を求めることが可能になる．さらに，大域探索ソルバとして既存の制約プログラミングシステムを用いるため，問題を記述するだけで適用可能であり，プログラムを書く必要が無い．評価実験によって，提案手法によるハイブリッドソルバが，clp(FD) を含む既存の大域探索ソルバよりも良い解を早く発見し得ることが示された．

Global search and local search are two of major approaches to combinatorial optimization problems. We propose here a new hybrid optimization method of these two approaches. The method (1) divides the search space into small subspaces using disjunctive constraints contained in the problem, (2) wanders from subspace to subspace, (3) optimizes each subspace visited using a global search solver. With the method, one can apply (pseudo) local search to any combinatorial optimization problems which contain disjunctive constraints using only a global search solver. More over, by using constraint programming systems as those global search solvers, one can apply the method just describing the problem without programming. Experimental results showed that the hybrid solver can find better solutions more quickly than existing global search solvers such as clp(FD).

1　はじめに

困難な組合せ最適化問題に対するアプローチとして，山登り法，焼きなまし法，遺伝的アルゴリズム等をはじめとする局所探索法の有効性が経験的に知られている．局所探索法は，(1) 現在の解を少しだけ変更した新しい解 (近傍) を複数作る，(2) 生成された近傍の中から次の解を選ぶ，というステップを繰り返し，解空間上を移動してゆくことでより良い解を求める手法である．近傍を生成する手続きは「近傍関数」と呼ばれる．局所探索法を適用するにはこの近傍関数が不可欠であるが，近傍関数は問題の種類ごとに異なるうえに，適切な近傍関数を設計するには問題に関する詳細な知識が必要とされる．結果として，局所探索のプログラムは作るのが難しく，作られたプログラムも問題に特化したものになる．

組合せ最適化問題に対する異なるアプローチとして制約プログラミングの技術が着目され，様々な制約プログラミングシステム [3] [4] [5] が開発されている．制約プログラミングの利点は，汎用性・柔軟性・簡便性である．プログラマは解きたい問題を制約条件の形で記述してソルバに渡すだけでよいし，その制約システムが提供する制約ライブラリを用いて記述した問題ならばどのような問題でも解かせることができる．しかし，従来の制約システムでソルバが用いるのは分枝限定法と深さ優先探索を組み合わせた大域探索で

A Hybrid Optimization Method for Combinatorial Optimization Problems with Disjunctive Constraints.

Shuji Ohnishi, 神戸大学大学院自然科学研究科, Graduate School of Science and Technology, Kobe University.

Naoyuki Tamura, 神戸大学学術情報基盤センター, Information Science and Technology Center, Kobe University.

コンピュータソフトウェア, Vol.22, No.3 (2005), pp.179–185.

[小論文] 2004 年 8 月 11 日受付.

あるため，探索効率の点で問題があった．

本研究の目的は，制約プログラミングシステムで局所探索を利用可能にすることである．つまり，問題の記述を与えるだけで，近傍関数を設計・プログラムすることなく局所探索を適用できるようにしたい．そのために，近傍を制約システム側で自動生成する方法を提案する．具体的には，問題の制約条件に含まれる論理的 OR に着目し，$p \vee q$ のうちどちらの条件を真にするかという選択をそれぞれの OR について切り替えることで，新しい解を作り出す．その際，既存の制約プログラミングシステムの大域探索機能をサブシステムとして利用する．

後で詳しく述べるが，この手法は，(1) 解空間全体を OR 制約を用いて小空間に分割し，(2) 訪れた小空間に対して大域探索を適用しながら，(3) 小空間から小空間へと局所探索的に移動していく，という，局所探索と大域探索を組み合わせたハイブリッド型の探索手法と見ることができる．

このように，提案手法は局所探索と大域探索の中間に位置するものであるが，それは探索効率の面でも同様である．人間によって入念に設計・コーディングされた局所探索ソルバは非常に高速である．したがって，解きたい問題についてすでに専用ソルバが存在する場合や，問題の性質が良く分かっていて自分で近傍関数をプログラムできる場合には，従来どおりその問題専用のソルバを使った方が良い．提案手法は，そのような専用ソルバが存在しない状況で，専用ソルバの作成という大きなプログラミングコストをかけずに大域探索よりも良い解を見付けたいという状況を想定している．

提案手法により，制約条件に OR を含むどのような問題に対しても，近傍関数をプログラムすることなく，問題を記述するだけで，局所探索を適用することが可能になる．また提案手法は，大域探索による最適化機能を持つような一般的な制約プログラミングシステムの上に，容易に導入することができる．評価実験では，提案手法によるハイブリッド型ソルバと clp(FD) を含む既存の大域探索ソルバとの比較を行う．

2 OR 制約を利用した近傍の自動生成

局所探索では，現在の解から次の解 (近傍) へと移動を繰り返す．問題を記述するだけで局所探索が使えるようにするには，この近傍生成ステップを自動化する必要がある．

しかし，近傍を自動生成するのに，通常の (人手による) 局所探索プログラムの様に変数の値を直接変更する方法は適用できない．それは，個々の問題特有の性質を利用できないからである．生成された解は問題の制約条件を満たさなくてはならないが，どの変数の値をどの程度変えれば良いか分からないのである．ランダムに変更したとしても，制約条件を満たさない解ばかりが生成されるだろう．

組合せ最適化問題は，目的関数 f と制約条件 C から成る．提案手法では，問題の制約条件 C に含まれる論理的 OR を利用して近傍を生成する．

2.1 簡単な近傍生成の例

提案手法による近傍の自動生成を，簡単な例で説明する．問題の制約条件 C が

$$(x{+}1 \leq y \vee y{+}2 \leq x) \wedge (z{+}3 \leq w \vee w{+}4 \leq z) \quad (1)$$

だったとする (目的関数 f は何でも良い)．このような形の制約条件は，スケジューリング問題やカッティング問題で頻繁に現れる．そして，現在の解が

$$(x, y, z, w) = (1, 3, 2, 5) \quad (2)$$

だったとする．この時，解 (2) は，

- $(x + 1 \leq y \vee y + 2 \leq x)$ の $(x + 1 \leq y)$ と，
- $(z + 3 \leq w \vee w + 4 \leq z)$ の $(z + 3 \leq w)$

を満たしている．したがって，解 (2) は制約 (1) を満たすだけではなく，より厳しい制約である

$$x + 1 \leq y \wedge z + 3 \leq w \quad (3)$$

の解にもなっている．

ここで，(3) の $(x + 1 \leq y)$ を OR で結ばれたもう一方の制約 $(y + 2 \leq x)$ と入れ替えて，新しい制約条件

$$y + 2 \leq x \wedge z + 3 \leq w \quad (4)$$

を作る．そして，既存の大域探索ソルバを用いて，この制約条件の下で目的関数 f を最適化する．そうして求まった解を，近傍解のひとつとする．

同様に，(3) の $(z+3 \leq w)$ を $(w+4 \leq z)$ と入れ替えて，

$$x+1 \leq y \wedge w+4 \leq z \qquad (5)$$

という制約条件をつくり，この条件のもとで f を最適化して，得られた解をふたつめの近傍解とする．

(4) も (5) も制約条件 (1) より厳しい制約条件であるから，得られたふたつの解は確かに元の問題の解として正しい．このように，制約条件に含まれる OR 制約 $(p \vee q)$ に着目し，現在の解が満たしている制約 p をもう一方の q に切り替え，大域探索ソルバに最適化させることで，新しい近傍解を生成する．

2.2 制約条件の AND-OR 構造とその OR-AND 構造への変換

上の例のように，制約条件は通常複数の制約の AND 結合から成る．さらに，個々の制約は論理的 OR を含んでいることが多い．例えば $y \neq z$ は明示的には OR を含まないが，$y < z \vee z < y$ と同値であり，暗黙的に OR を含んでいる．このように，組合せ最適化問題の制約条件は，OR を AND で結合した AND-OR 構造をしているか，またはそのような構造に変換できることが多い．

以下では,「OR 結合」と言った場合には排他的であることを前提にする．これは，探索の重複を避けるためである．任意の OR 結合 $p \vee q$ は，同値な排他的 OR 結合 $p \vee (\neg p \wedge q)$ に変換することができる．

問題の制約条件 C が AND-OR 構造を持つとき，すなわち $C = C_1 \wedge \cdots \wedge C_n$ で $C_i = c_{i,1} \vee \cdots \vee c_{i,m_i}$ であるとき，制約条件 C は次のような OR-AND 構造に変換できる:

$$\begin{aligned} C = &(c_{1,1} \wedge c_{2,1} \wedge \cdots \wedge c_{n,1}) \\ &\vee (c_{1,2} \wedge c_{2,1} \wedge \cdots \wedge c_{n,1}) \\ &\cdots \\ &\vee (c_{1,m_1} \wedge c_{2,m_2} \wedge \cdots \wedge c_{n,m_n}). \end{aligned} \qquad (6)$$

この OR-AND 構造の，OR で結ばれた各制約は互いに排他的であり，C を満たす解はそれらの内いずれかちょうどひとつを満たしている．

現在の解が満たしているのが $(c_{1,1} \wedge \cdots \wedge c_{n,1})$ だとする．この制約から，$c_{i,1}$ をひとつ選び，それを $c_{i,j}(2 \leq j \leq m_i)$ のひとつと入れ替えて新しい制約を作る．そして，その制約の下で目的関数 f を最適化し，近傍解のひとつとする．したがって，ひとつの解から mn 個の近傍解を作り出すことができる．

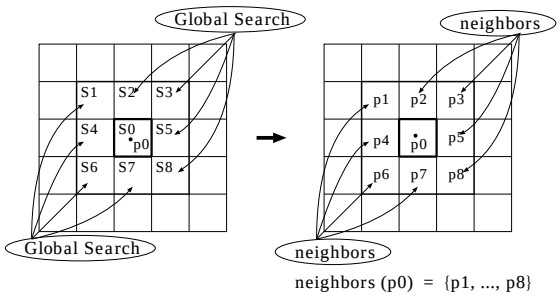

図 1 探索空間の分割と近傍の生成

2.3 解空間から見た提案手法の意味

提案手法の意味を，解空間で考えてみる．難しい組合せ最適化問題に対して大域探索で良い解が見つけられないのは，解空間が大きすぎるからである．解空間が十分に小さければ，大域探索により妥当な時間で確実に最適解を求めることができる．

そこで，解空間全体を大域探索が可能な大きさにまで細かく分割する．そして，次のように近傍を生成する (図 1).

1. 現在の解 p_0 が，小空間 S_0 に含まれるとする．
2. その周囲の小空間 $S_1, \cdots, S_n$ に対してそれぞれ大域探索を適用し，最適解 $p_1, \cdots, p_n$ を得る (実行可能解をひとつも持たない小空間は無視する).
3. $\{p_1, \cdots, p_n\}$ を p_0 の近傍とする．

提案手法では，制約条件の AND-OR 結合を OR-AND 結合に変換することで探索空間を分割し，現在の解が満たしている制約をひとつだけ入れ替えることで「周囲の小空間」を作り出しているのである．制約の数が n で，個々の制約が m 個の制約の OR 結合に分解されるとき，解空間は m^n 個という非常に大きな数の小空間に分解されることになる．

このように考えると，提案手法は，近傍を自動生成しているというよりも局所探索と大域探索を組み合わせたハイブリッド型の探索手法と見ることができる．解空間を分割し，分割された小空間の間の近傍関係を定義するために OR 制約を利用しているのである．

3 近傍のサイズの制御

制約が互いに密接な関係を持つ場合，制約を単純にひとつだけ入れ替える方法には問題がある．

簡単な例で説明する．制約条件が $C = (a \vee b) \wedge (a \vee c)$ だったとする．C を OR-AND 構造に変換すると，次のようになる．

$$C = (a \wedge a) \vee (a \wedge c) \vee (b \wedge c) \vee (b \wedge a). \quad (7)$$

現在の解がこれらの内で $(a \wedge a)$ を満たしているとする．すると，近傍となる制約は $b \wedge a$(左の a を入れ替えた) と $a \wedge c$(右の a を入れ替えた) である．

ここで注意しなくてはいけないのは，元の制約条件に含まれる OR が排他的であること，つまり $a \wedge b = false$, $a \wedge c = false$ という点である．したがって，制約をひとつだけ入れ替えても矛盾が発生してしまって近傍解を得ることができない．もしふたつ同時に入れ替えることが許されるなら，$(b \wedge c)$ という近傍があり得る．

しかし，ふたつ以上同時に入れ替えることを許すと，今度は組合せ爆発が起こって近傍が大きくなりすぎてしまう．近傍は必要最小限に保つのが良い．したがって，矛盾を避けつつ変更する制約の数を抑える方法が必要になる．

そのために，制約間での変数の共有関係を利用する．制約の中には，同じ変数を共有しているものがある．そのような制約同士は当然相互に影響しあっているから，近傍を作る際にそのうちの一方だけを変更したとしても，もう一方の制約に違反してしまう可能性が高い．よって，変数を共有している制約をグループにまとめ，そのグループ間ではふたつ以上の変更を許すことにする．

ここでも簡単な例で説明する．いま，制約条件が $C = C_1 \wedge C_2 \wedge C_3$ で，各 C_i について $C_i = c_{i,1} \vee c_{i,2} \vee c_{i,3}$ であり，C_1 と C_2 が変数を共有しているとする．現在の解は $(c_{11} \wedge c_{21} \wedge c_{31})$ を満たしているとする．このとき，ひとつだけ制約を変更することによってできる近傍は次のようになる:

$$\begin{cases} (c_{12} \wedge c_{21} \wedge c_{31}), \\ (c_{13} \wedge c_{21} \wedge c_{31}), \\ (c_{11} \wedge c_{22} \wedge c_{31}), \\ (c_{11} \wedge c_{23} \wedge c_{31}), \\ (c_{11} \wedge c_{21} \wedge c_{32}), \\ (c_{11} \wedge c_{21} \wedge c_{33}). \end{cases}$$

これに対し，変数を共有している C_1 と C_2 については両方同時に変更してよいとすると，近傍には次の要素が追加される:

$$\begin{cases} (c_{12} \wedge c_{22} \wedge c_{31}), \\ (c_{12} \wedge c_{23} \wedge c_{31}), \\ (c_{13} \wedge c_{22} \wedge c_{31}), \\ (c_{13} \wedge c_{23} \wedge c_{31}). \end{cases}$$

したがって近傍のサイズは，従来どおりひとつしか変更を許さない場合は 6, ここで提案したように変数を共有している制約についてはいくつ変えても良いとすると 10 になる．全ての制約をいくつ入れ替えても良いとすると，近傍のサイズは $3^3 - 1 = 26$ にもなる．変数の共有関係を利用する方法が，矛盾を避けつつ爆発を抑制できることが分かる．

しかしこの方法は，全ての制約が同じ変数を共有している場合には，制約をいくつ入れ替えても良いとする方法と同じになってしまい，近傍サイズの爆発を抑えられない．経験的にそのような状況は少ないが，もしそうなった場合には，変更する制約の数を動的に変更するという方法がある．すなわち，始めは変更する制約の数を 1 としておき，近傍が小さすぎたり空になってしまったりするようなら徐々に増やす，という方法である．このふたつを組み合わせれば，近傍の大きさをできる限り小さく保つことができる．

4 評価実験

Cream[†1] は，筆者らの開発している整数制約解消のための Java クラスライブラリであり，一般的な制約プログラミングシステムと同じく，分枝限定法と深さ優先の大域探索による最適化機能を持つ．この Cream 上に，提案手法にしたがって近傍関数の自動

†1 http://bach.istc.kobe-u.ac.jp/cream/

生成機能を組み込んだ．近傍から次の解を選択する手法としては最も単純な山登り法を実装した．そうしてできたハイブリッド型ソルバと，Cream 既存の大域探索ソルバの性能を，NP 困難な問題のひとつである Non-guillotine Cutting Problem(NGCP) [2] で比較した．また，制約論理プログラミングシステムとして著名な GNU Prolog[†2] の clp(FD) 実装との比較も行った．clp(FD) も大域探索ソルバである．

4.1 Non-guillotine Cutting Problem

この問題では，一枚の大きな矩形の板から，同じく矩形の小さな板をいくつか切り出す．切り出す矩形にはいくつかの種類があり，それぞれ形 (長さと幅)，一枚あたりの価値，最低限切り出さなければならない数，最大限切り出せる数が指定されている．目的は，切り出した矩形の価値の合計を最大化するように，切り出す板を選び，切り出す位置を決定することである．

細かな条件を除けば，この問題の制約条件は「切り出す矩形は重なってはならない」ということである．「板 i が切り出される」ことを p_i と表し，「板 i と板 j が重ならない」ことを $non\_overlap(i,j)$ と表すと，問題の制約条件は「すべての板 i, j の組 $(i \neq j)$ について

$$(p_i \wedge p_j) \Rightarrow non\_overlap(i,j) \qquad (8)$$

が成り立つ」と表される．

4.2 提案手法により生成される近傍

$a \Rightarrow b$ は $\neg a \vee b$ と同値であるから，上の制約条件 (8) は $\neg(p_i \wedge p_j) \vee non\_overlap(i,j)$ と同値である．さらにこれを排他的 OR に変更して，

$$\neg(p_i \wedge p_j) \vee (p_i \wedge p_j \wedge non\_overlap(i,j)) \qquad (9)$$

となる．これで，提案手法を適用して近傍を生成することができる (この変換処理は Cream が自動的に行う)．制約 (9) は，$\neg(p_i \wedge p_j)$ と $(p_i \wedge p_j \wedge non\_overlap(i,j))$ というふたつの制約に分割される．前者は板 i と板 j の内少なくともどちらか一方は切り出されないことを意味する．後者は二枚の板がともに切り出され，お互いに重複しないことを意味する．二枚の板の組合せごとにこのふたつの制約を切り替えることで近傍が作成される．したがってこの近傍関数は，それぞれの板について切り出す/切り出さないを切り替えていると見ることができる (前節の，変数を共有している制約に関する処理も実行される)．

表 1 実験結果

問題		既存の大域探索		提案手法
番号	最適値	clp(FD) 大域探索	Cream 大域探索	Hybrid 山登り法
1	164	opt(0.64)	opt(267)	opt(66.3)
2	230	219	146	opt(18.6)
3	358	231	268	opt(87.9)
4	924	653	288	764

4.3 実験結果

NGCP のインスタンスは，OR-Library [1] から取った[†3]．実験結果を表 1 に示す．表中の $opt(T)$ は最適解に T 秒で到達したということを表している．300 秒以内に最適解を発見できない場合は打ち切りとし，300 秒以内に発見できた最も良い解の値を記した．

問題 1 は最も小さく簡単である．全てのソルバが最適解を発見した．clp(FD) は非常に短い時間で最適解を発見している．問題 2 と 3 は，1 に比べて少し大きい問題である．既存の大域探索ソルバ (“clp(FD) 大域探索” と “Cream 大域探索”) はどちらも最適解を発見できず，提案手法によるハイブリッド型ソルバ (“Hybrid 山登り法”) は最適解を発見した．最も大きく難しい問題 4 については，どのソルバも最適解は発見できなかった．しかし，300 秒以内に発見できた最も良い解を比較すると，clp(FD) の 653 に対して，ハイブリッド型ソルバは 764 というより良い解を発見している．

結果として，提案手法により実装されたハイブリッド型ソルバは，全ての問題について clp(FD) よりも良い解を発見することができた．ここで注目すべきなのは，ハイブリッドソルバが大域探索ソルバと同じく

†2 http://pauillac.inria.fr/~diaz/gnu-prolog/

†3 http://mscmga.ms.ic.ac.uk/jeb/pub/ngcutap.txt

問題記述のみを使用していることである．問題記述のみを利用して近傍を生成し，それを用いて局所探索することで，大域探索よりも良い解を早く発見できている．ハイブリッドソルバを利用するために余分なプログラミングは必要無い．

4.4 専用の局所探索アルゴリズムとの比較

Beasley は [2] で NGCP に対する遺伝的アルゴリズムを用いた最適化手法を提案し，表 1 と同じ問題を使って評価している．その結果によると，Beasley の専用局所探索プログラムは，四つの問題いずれについても最適解を発見している．かかった時間も，問題 1 から 3 については 0.1–0.2 秒，問題 4 については 2 秒程度である．実験環境の違い (本研究の 1.7GHz CPU に対して Beasley の 225MHz) も考えると，専用の局所探索ソルバがいかに速いかということが分かる．

5 まとめと今後の課題

局所探索と大域探索を組み合わせた新しいハイブリッド型最適化手法を提案した．提案手法では，組合せ最適化問題の制約条件に含まれる論理的 OR を利用して，局所探索を適用する際に必要な近傍解を自動生成する．提案手法により，制約条件に OR を含むどのような問題に対しても，近傍関数をプログラムすることなく問題を記述するだけで局所探索を適用することが可能になる．また評価実験において，提案手法によるハイブリッド型ソルバが，clp(FD) を含む既存の大域探索ソルバよりも良い解を早く発見し得ることが確かめられた．

プログラミングの必要が無いという大きな利点があるとはいえ，人間によって特化された専用局所探索ソルバに比べれば，提案手法の探索性能はまだまだ不十分である．したがって探索性能の向上が今後の課題である．例えば次のような問題と対策が考えられる．

まず，現在の近傍生成法は制約条件 (の構造) から静的に決まってしまう．実行状況を動的に利用することで，より適切な近傍が生成できるはずである．例えば近傍生成において矛盾が生じてしまった場合，“制約 P を切り替えたら制約 Q と矛盾した”，という記録を残しておけば，次に制約 P を切り替えるときには Q の方も切り替える，という対策がとれる．また，度々矛盾を生じるような制約がある場合，その制約は問題の最も重要な制約 (ボトルネック) であると考えられるので，その制約は分割せずに置いておく，といったことも可能である．このようにすれば，変数の共有関係だけを利用するよりもより問題に適したきめの細かい近傍生成が可能であろう．

次に，現在の手法は目的関数を全く考慮していない．簡単な例だが，例えば $z = x - y$ の最大化が目的であった場合，x の値を増やす (y の値を減らす) ような切替えを優先的に実行した方が良いはずである．

また，生成された近傍から次の解を選ぶ手法として，現在のところ最も単純な山登り法しか実装されていない．焼きなまし法やタブーサーチ，遺伝的アルゴリズムといった，山登り法より有効とされている手法を導入する必要があるだろう．ただし，それらの手法の中には近傍の他に付加的なデータ構造やアルゴリズムを必要とするものがある．タブーサーチのタブーリストや，遺伝的アルゴリズムの交配と突然変異などがその例である．従来の局所探索とは異なる点の多い提案手法において，それらの要素をどのように実現するかという課題がある．

もちろん，これらの改良手法を導入することによるオーバーヘッドの増加も考慮する必要がある．

最後に，有益な助言をいただいた査読者の方々に感謝いたします．

参 考 文 献

[1] Beasley, J. E.: OR-Library: Distributing Test Problems by Electronic Mail, *Journal of the Operational Research Society*, Vol. 41, No. 11(1990), pp. 1069–1072.

[2] Beasley, J. E.: A Population Heuristic for Constrained Two-Dimensional Non-Guillotine Cutting, July 19 2000.

[3] Codognet, P. and Diaz, D.: Compiling Constraints in clp(FD), *Journal of Logic Programming*, Vol. 27, No. 3(1996), pp. 185–226.

[4] Puget, J.-F. and Leconte, M.: Beyond the glass-box: Constraints as objects, in *Proceedings of the International Logic Programming Symposium* (Lloyd, J. W.(ed.)), Portland, Oregon, MIT Press, 1995, pp. 513–527.

[5] SICS: *SICStus User Manual. Version 3.10.0*, Swedish Institute of Computer Science, 2002.