数理解析研究所講究録
1114巻 1999年 211-220

集合被覆問題に対する局所探索法について

京都大学 情報学研究科 *岸田 正博 KISHIDA Masahiro
京都大学 情報学研究科 柳浦 睦憲 YAGIURA Mutsunori
京都大学 情報学研究科 茨木 俊秀 IBARAKI Toshihide

1 はじめに

集合被覆問題 (SCP) は代表的な組合せ最適化問題の1つであり, NP困難であることが知られている. これは, 与えられた要素集合を全てカバーする集合族の中で重み最小のものを求めるという問題である. この問題は, バス運転手, パイロット, 鉄道員などのスケジューリング [1][5][10], 施設の配置 [11], データの論理的解析 [4] 等様々な実際問題への応用がある.

この問題に対しては様々な解法が提案されている. 厳密解法の分野では, Fisher と Kedia が双対問題を利用した解法を与え, $m = 200$, $n = 2000$ までの問題に適用している [8]. Beasley はラグランジュ緩和問題を利用したアルゴリズムに Gomory f-カットを適用して $m = 400$, $n = 4000$ までの問題例を解いている [2]. しかし, 大規模な問題例を解くためには近似解法が必要になってくる. 代表的なものとして, Beasley と Chu による遺伝アルゴリズム [3], Jacobs と Brusco によるアニーリング法 [9] などがあり, $m = 1000$, $n = 10000$ までの問題例に対して優れた近似解を得ている. また, Caprara, Fischetti, Toth や Ceria, Nobil, Sassano はラグランジュ乗数を用いて再評価した重みに対して欲張り法を適用する方法で $m \simeq 5000$, $n \simeq 1000000$ という非常に大規模な問題例に対しても効果的なアルゴリズムを提案している [5][6].

本研究では, 同時に3つまでの集合を出し入れすることによって得られる解集合を近傍とする局所探索法を提案する. これは, 従来のアルゴリズムに比べ大きな近傍となっているが, 単純に近傍を広げただけでは効率が悪くなるので, 大きな近傍を扱う際には, 解の質を悪くすることなく効率的に近傍を探索する工夫を行っている. また, 実行不可能解も許すことによって, 探索の柔軟性を高めている. さらに, 大規模な問題例に対応するため, ラグランジュ緩和問題を利用して, 効果的な問題サイズの縮小を行っている. 代表的なベンチマーク問題に対する計算実験より, 本研究で提案したアルゴリズムは他のアルゴリズムと比べて, 良質の解が得られることが確認された.

2 集合被覆問題

集合被覆問題とは, 要素集合 $M=\{1,...,m\}$ と, Mの部分集合族 S_j, $j \in N=\{1,...,n\}$ が与えられたとき, Mの全ての要素をカバーするようにいくつかの集合を選び, 選んだ集合に付けられた重みの総和を最小にする問題である. 0-1 変数 $\boldsymbol{x} \in \{0,1\}^n$ を用いると, この問題は以下のように定式化される.

$$\text{minimize} \quad cost(\boldsymbol{x}) = \sum_{j \in N} c_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j \in N} a_{ij} x_j \geq 1, \ i \in M \tag{2}$$

$$x_j \in \{0,1\}, \ j \in N \tag{3}$$

a_{ij}: 集合 S_jが要素 i をカバーするなら 1, そうでなければ 0

c_j: 集合 S_jの重み (c_jは正整数と仮定する)

x_j: 集合 S_jが解に含まれるなら 1, そうでなければ 0

3 ラグランジュ緩和問題の利用

集合被覆問題に対しては, ラグランジュ緩和問題を解くことによって得られる情報が非常に有用であることが知られており, 多くのアルゴリズムにおいて利用されている. 本研究では, ラグランジュ緩和問題から得られた情報を用いて, 最適解に含まれる可能性が高いと思われる集合だけを取り出すことにより問題サイズを縮小している. これは, 集合数 n が非常に大きい場合, 解の質の向上とアルゴリズムの高速化の両面において有効である.

3.1 ラグランジュ緩和問題と劣勾配法

制約条件 (2) に関するラグランジュ乗数ベクトル $\boldsymbol{v}\in R_+^m$に対し,

$$c_j(\boldsymbol{v}) = c_j - \sum_{i\in S_j} v_i$$

は, 集合 S_jに関する相対コストと呼ばれる. これらを用いて, ラグランジュ緩和問題は

$$L(\boldsymbol{v}) = \min_{\boldsymbol{x}} \sum_{j\in N} c_j(\boldsymbol{v})x_j + \sum_{i\in M} v_i \tag{4}$$

$$\text{subject to} \quad x_j \in \{0,1\}, \quad j \in N \tag{5}$$

と書かれる. 任意の $\boldsymbol{v}\in R_+^m$に対し, ラグランジュ緩和問題の最適値 $L(\boldsymbol{v})$ は, 集合被覆問題の最適解 $\boldsymbol{x}^*$に対して必ず $L(\boldsymbol{v}) \leq c\boldsymbol{x}^*$を満たし, 集合被覆問題の下界値を与える. ラグランジュ双対問題は, 下界値 $L(\boldsymbol{v})$ を最大化するラグランジュ乗数ベクトル $\boldsymbol{v}^*$を見つける問題である. $\boldsymbol{v}$を定めると, (4)-(5) 式の最適解 $\boldsymbol{x}(\boldsymbol{v})$ は, $c_j(\boldsymbol{v}) < 0$ ならば $x_j(\boldsymbol{v})$=1, $c_j(\boldsymbol{v}) > 0$ ならば $x_j(\boldsymbol{v})$=0, $c_j(\boldsymbol{v})$=0 ならば $x_j(\boldsymbol{v})\in \{0,1\}$ で与えられる. (4)-(5) 式に整数性 (integrality property) があるため, 集合被覆問題の LP 緩和問題の双対問題, すなわち,

$$\max\left\{\sum_{i\in M} v_i \;\middle|\; \sum_{i\in S_j} v_i \leq c_j \ (j\in N),\ v_i \geq 0 \ (i\in M)\right\}$$

の任意の最適解はラグランジュ双対問題の最適解でもある. しかし, 最適なラグランジュ乗数ベクトル $\boldsymbol{v}^*$を計算するのは, 大規模な問題に対し非常に計算時間がかかる. そのため, 通常, 短い計算時間で近似的にラグランジュ乗数ベクトルを求めるために劣勾配法が利用される.

劣勾配ベクトル $\boldsymbol{s}(\boldsymbol{v})$ は, ベクトル $\boldsymbol{v}$に対し,

$$s_i(\boldsymbol{v}) = 1 - \sum_{j\in N} a_{ij}x_j(\boldsymbol{v}), \quad i \in M \tag{6}$$

で与えられる. 劣勾配法は, 次式

$$v_i^{k+1} = \max\left\{v_i^k + \lambda\frac{UB - L(\boldsymbol{v}^k)}{||\boldsymbol{s}(\boldsymbol{v}^k)||^2}s_i(\boldsymbol{v}^k),\ 0\right\}, \quad i\in M \tag{7}$$

によって非負のラグランジュ乗数ベクトル $\boldsymbol{v}^0, \boldsymbol{v}^1, \ldots$を生成し, $L(\boldsymbol{v}^k)$ が最大となる $\boldsymbol{v}^k$を $\boldsymbol{v}^*$の近似解 $\boldsymbol{v}^\diamond$として利用する方法である. UBは集合被覆問題の目的関数の上限値で, $\lambda > 0$ は収束の速度を決めるパラメータである.

初期値$\boldsymbol{v}^0$の定め方, およびλの更新方法には色々あるが, ここでは, 文献 [6] を参考に以下のように行った. ラグランジュ乗数ベクトルの初期値$\boldsymbol{v}^0$は,

$$v_i^0 := \min\left\{ \frac{c_j}{|S_j|} \;\middle|\; i \in S_j \right\} \tag{8}$$

で与える. λは, 初期値を$\lambda = \lambda_0$とし, 現在得られている下界値の最良値がβ回毎の反復で更新されなければ, $\lambda := \lambda/\rho$とする. 劣勾配法の反復は, $\lambda < \lambda_{min}$となった時点で終了する. 計算実験では, $\lambda_0 = 4$, $\beta = 15$, $\rho = 1.2$, $\lambda_{min} = 0.002$ と設定した.

UBは以下のような欲張り法を適用して求める. $\boldsymbol{x}$=$\mathbf{0}$ から始め, 各集合に対し,

$$\begin{aligned} u_j(\boldsymbol{x}) &= \left| \left\{ i \in S_j \;\middle|\; \sum_{h \in N} a_{ih} x_h = 0 \right\} \right| \\ r_j(\boldsymbol{x}) &= \frac{c_j}{u_j(\boldsymbol{x})} \end{aligned}$$

とするとき, $r_j(\boldsymbol{x})$ の値が最小の集合S_jを解に加える (すなわち $x_j := 1$ とする) という操作を, 全ての要素がカバーされるまで繰り返す.

3.2 問題サイズの縮小

問題サイズの縮小は, n 個の集合S_jの中から, 有効と考えられる一部の集合を残すことによって構成される. 本研究では以下の方法により, 探索の対象となる集合を選択した.

1. $N' := \{\, j \in N \mid c_j(\boldsymbol{v}^\diamond) \leq \gamma \,\}$ (γは実数値を取るパラメータ) とする.
2. ステップ 1 で求めた $|N'|$ に対して $|N'| > 5m$ が成り立つ場合は, 相対コスト $c_j(\boldsymbol{v}^\diamond)$ の小さい順に $5m$ 個の集合を選び, これを改めて N'とする.
3. 各要素 i に対し, i を含む集合を相対コスト $c_j(\boldsymbol{v}^\diamond)$ の小さい順に 5 個選び, N'に加える.

局所探索法においては, 選ばれた集合族 N'のみを探索の対象とし, $\forall j \in N - N'$ に対しては, $x_j = 0$ に固定する.

4 局所探索法

局所探索法とは, 現在の解 $\boldsymbol{x}$ の近傍 $NB(\boldsymbol{x})$ 内に $\boldsymbol{x}$ より良い解があればそれに置き換える, という操作を, 近傍内に改善解がなくなるまで反復する方法である. 近傍 $NB(\boldsymbol{x})$ は解 $\boldsymbol{x}$ に多少の変更を加えて得られる解集合である. 近傍内により良い解が存在しない解を局所最適解と呼ぶ. 局所探索法の動作は, 探索空間, 解の評価関数, および近傍により決定される. 以下では, これらの詳細について述べる.

4.1 探索空間と解の評価関数

本研究で提案する局所探索では, 探索空間は任意の $\boldsymbol{x} \in \{0,1\}^n$とする. すなわち, 集合被覆問題の制約条件 (2) を満たさない実行不可能解も探索空間に含める. そこで, 制約条件 (2) を考慮したペナルティ関数を以下のように定義し, 評価関数に用いる. 各要素 i に対する制約条件 (2) の制約違反を表す関数$\theta_i(\boldsymbol{x})$ を,

$$\theta_i(\boldsymbol{x}) = \max\left\{ 1 - \sum_{j \in N} a_{ij} x_j,\; 0 \right\},$$

ペナルティ係数を $p_i(>0)$ とし, 解の評価関数を,

$$pcost(\boldsymbol{x}) = \sum_{j\in N} c_j x_j + \sum_{i\in M} p_i\theta_i(\boldsymbol{x}) \tag{9}$$

と定義する. ペナルティ係数 p_i は, 探索の状況に応じて動的に変化させる. これについては 4.3 節で述べる.

4.2 近傍と探索

$\boldsymbol{x}$ と $\boldsymbol{x}'\in\{0,\ 1\}^n$ のハミング距離を

$$d(\boldsymbol{x},\boldsymbol{x}') = \left|\{j\in N|\ x_j\neq x'_j\}\right|$$

で表し, $\boldsymbol{x}$ の近傍 $NB_h(\boldsymbol{x})$ を

$$NB_h(\boldsymbol{x}) = \{\boldsymbol{x}'\in\{0,1\}^n|\ d(\boldsymbol{x},\boldsymbol{x}')\leq h\}$$

と定義する. すなわち, $\boldsymbol{x}$ からのハミング距離が h 以下の解集合である. 本研究では, $h\leq 3$ の近傍に基づく局所探索を行う. 以下では, $h(\leq 3)$ に対し, $NB_h(\boldsymbol{x})$ 内に改善解を見つけてそれに移動するか, または改善解がないことを結論するまでの手順を説明し, それに必要な時間を考察する. 説明の都合上,

$$\begin{aligned} n' &= \sum_{j\in N} x_j \\ t &= \max\left\{\sum_{i\in M} a_{ij}\ \middle|\ j\in N\right\} \\ l &= \max\left\{\sum_{j\in N} a_{ij}\ \middle|\ i\in M\right\} \end{aligned}$$

を定義しておく.

近傍を効率的に探索するため, $h\geq 2$ に対しては, $NB_{h-1}(\boldsymbol{x})$ に改善解がない時に限り $NB_h(\boldsymbol{x})$ を調べる. なお, 証明は省くが, 以下の探索方法により $NB_h(\boldsymbol{x})$ 内に $pcost(\boldsymbol{x})$ を改善する解が存在すれば, そのうちの 1 つを必ず発見できることが示せる.

$d(\boldsymbol{x},\boldsymbol{x}')$=1 であるような改善解は, 現在の解から 1 つの 0-1 変数の値を反転させたものである. 各変数 x_j に対し, その変数の値を反転させたとき, 改善がおこるかどうかを $O(1)$ 時間で判定するため, その変数の値を反転させたときの評価関数 $pcost(\boldsymbol{x})$ の変化量をメモリーに記憶しておく. その結果, 調べるべき集合の候補が $O(n)$ 個, 改善解が見つかったときのメモリーの更新に $O(tl)$ 時間かかるので, この探索は $O(n+tl)$ の計算時間で実行できる.

$d(\boldsymbol{x},\boldsymbol{x}')$=2 であるような解は, 現在の解から 1 つの変数を 1 から 0 に, 1 つの変数を 0 から 1 に反転させたものを考えれば十分である. $x_j=1$ であるような $O(n')$ 個の各変数に対してその値を反転させたとき, $x_j=0$ であるような変数の中で, その値を 1 に反転することによって改善がおこるような候補を $O(tl)$ 時間で見つけることができる. したがって, 全体の手間は $O(n'tl)$ 時間である.

$d(\boldsymbol{x},\boldsymbol{x}')$=3 であるような解は, 現在の解から 2 つの変数を 1 から 0 に, 1 つの変数を 0 から 1 に反転させたものと, 1 つの変数を 1 から 0 に, 2 つの変数を 0 から 1 に反転させたものの 2 種類ある. $x_j=1$ であるような $O(n')$ 個の各変数に対してその値を反転させたとき, $x_j=0$ であるような変数の中で, その値を 1 に反転することによって改善解が得られる可能性のある候補の数が $O(\min\{n,tl\})$ 個, 各候補の変数の値を反転させたとき, さらにもう 1 つの変数の値を反転させることによって改善がおこるような候補を $O(tl)$ 時間で見つけることができる. したがって, 全体の手間は $O(n'tl\ \min\{n,tl\})$ 時間である.

結局, $NB_3(\boldsymbol{x})$ 全体の探索を $O(n'tl\ \min\{n,tl\})$ 時間で行えることが分かった. $NB_3(\boldsymbol{x})$ の探索をこのような工夫なしに行うと, 通常 $O(tn^3)$ 時間必要となるが, $n' \leq n$, $l \leq n$ であることから十分な高速化がされている. 特に, $l \ll n$ である問題例に対しては, かなりの高速化が期待できる.

4.3 ペナルティ係数の更新

通常の局所探索法は, 局所最適解に到達した時点で探索を終了し, その時点の暫定解を出力する. しかし, 局所探索法を1度行っただけでは未探索の領域にさらによい解が隠れているという危惧が残る. また, 本研究で提案している局所探索法は探索空間に実行不可能解も含めているため, ペナルティ係数 p_i の値を十分大きくしない限り1度の探索で必ず実行可能解が得られる保証はない. そこで, 局所探索法が局所最適解を求めて停止した時点で各要素のペナルティ係数 p_i を更新し, 前回の局所最適解を初期解としてさらに局所探索法を行うという方法をとる. ただし, ペナルティ係数は, 1. 解の質の向上, 2. 高い頻度で実行可能解を得る, という2点について十分考慮して設定する必要がある. まず, p_i の初期値は

$$p_i := \min\left\{ \frac{c_j}{|S_j|} \;\middle|\; i \in S_j \right\}$$

と定める. 次に, 現在の局所最適解 $\boldsymbol{x}$ と暫定値 UB に対して, $cost(\boldsymbol{x}) < UB - 1$ のときは,

$$p_i := p_i \left(1 + \theta_i(\boldsymbol{x})\ \max\left\{ \frac{UB - 1 - cost(\boldsymbol{x})}{UB},\ \Delta^+ \right\} \right),$$

$$\theta_i(\boldsymbol{x}) = \max\left\{ 1 - \sum_{j \in N} a_{ij} x_j,\ 0 \right\},$$

そうでないときは,

$$p_i := p_i \left(1 + \lambda_i(\boldsymbol{x})\ \min\left\{ \frac{UB - 1 - cost(\boldsymbol{x})}{UB},\ \Delta^- \right\} \right),$$

$$\lambda_i(\boldsymbol{x}) = \frac{\sum_{j \in N} a_{ij} x_j + 1}{\max_{i \in M} \sum_{j \in N} a_{ij} x_j + 1},$$

と更新する. ここで, Δ^+ と Δ^- は, $\Delta^+ > 0$, $-1 < \Delta^- < 0$ をみたすパラメータである.

$cost(\boldsymbol{x}) < UB - 1$ ならば, $\boldsymbol{x}$ は実行不可能解であるが (実行可能解であれば暫定値 UB が更新されるので不等式をみたさない), $\boldsymbol{x}$ に集合を追加して実行可能解を得ることで, 暫定解を更新する可能性があるので, カバーされてない要素のペナルティ係数を増加させる. 逆に, $cost(\boldsymbol{x}) \geq UB - 1$ ならば, $\boldsymbol{x}$ に集合を追加しても, 暫定解は更新できないので, 各要素のペナルティ係数を減少させる.

4.4 欲張り法の併用

前節で述べたペナルティ係数の更新ルールにより, 実行可能解が得られる可能性は高くなるが, 必ずしも十分な頻度で実行可能解を得ることはできない. しかし, 局所探索法で得られた局所最適解は, 多くの場合, 少しの変形で良質な実行可能解にできると考えられる. そこで, 本アルゴリズムでは, 得られた局所最適解に対して以下の欲張り法を適用し, 実行可能解を得る.

x_j の値を0から1に変化させたとき $pcost(\boldsymbol{x})$ の増加量を最小化するような集合 S_j を選び $x_j := 1$ とする操作を, 実行可能解が得られるまで繰り返す. さらに, 得られた解に対して, 実行可能解のみを探索空間とした, 近傍 $NB_h(\boldsymbol{x})(h \leq 3)$ に基づく局所探索法を適用する.

5 アルゴリズムの枠組

本研究で提案するアルゴリズムは, ラグランジュ緩和問題を解くことにより得られた情報を利用して問題サイズを縮小し, 縮小した問題に対して局所探索法を適用するというアルゴリズムで, 全体の枠組は以下のように記述できる.

アルゴリズム LS

1. 3.1 節の欲張り法を用いて実行可能解 $\boldsymbol{x}$ を求め, $UB := cost(\boldsymbol{x})$ とする.
2. 3.1 節の劣勾配法を用いてラグランジュ緩和問題を解き, ラグランジュ乗数ベクトル $\boldsymbol{v}^\diamond$を求める. また, $\boldsymbol{x}$:=**0** とする.
3. $\boldsymbol{v}^\diamond$に対する相対コスト $c_j(\boldsymbol{v}^\diamond)$ を計算する. これに基づき, 3.2 節のアルゴリズムにしたがって探索の対象とする集合を選び, それ以外の集合は $x_j := 0$ に固定する.
   $p_i := \min\left\{ \frac{c_j}{|S_j|} \;\middle|\; i \in S_j \right\}$とする.
4. 計算時間が制限時間 T (Tは実数値を取るパラメータ) に達したら, UBを出力してアルゴリズムを終了する.
5. 初期解を $\boldsymbol{x}$ として 4.1 節と 4.2 節に述べた局所探索法を行い, 局所最適解 $\boldsymbol{x}^*$を得る.
   $\boldsymbol{x} := \boldsymbol{x}^*$とする.
6. 局所探索法の探索中に実行可能解が得られた場合, その中で目的関数値の最小の値を UB'とする (実行可能解が得られなかった場合は $UB' = \infty$).
   $UB' < UB$ならば, $UB := UB'$とする.
7. 局所最適解 $\boldsymbol{x}^*$が実行不可能解である場合, その局所最適解に 4.4 節の欲張り法を適用して実行可能解 $\boldsymbol{x}^\diamond$を得る. 実行可能解 $\boldsymbol{x}^\diamond$に対し, 実行可能解のみを探索領域とした局所探索法を適用し, 得られた局所最適解を改めて $\boldsymbol{x}^\diamond$とおく. $cost(\boldsymbol{x}^\diamond) < UB$であれば, $UB := cost(\boldsymbol{x}^\diamond)$ とする.
8. 4.3 節の方法に従ってペナルティ係数 p_iを更新する.
   ステップ 4 へ戻る.

6 計算実験

実験は, ワークステーション Sun Ultra 2 Model 2300 (300 MHz, 1 GB memory) 上で C 言語を用いて行った. 用いた問題例は, Beasely の OR-Library (http://mscmga.ms.ic.ac.uk/jeb/orlib/scpinfo.html) からの代表的なベンチマーク問題で, 全て最適解が知られていない問題である. 問題例の詳細については, 表 1 に記してある. タイプ e~h はランダムに生成された問題である. 一方, rail はイタリアの鉄道会社のスケジューリング問題である.

6.1 近傍による比較

表 2 に, 近傍を NB_1, NB_2, NB_3とした局所探索法による結果を示す. 実験は, 各近傍とも, 各問題に対しそれぞれ 10 回ずつ行った. 表中の best known には既知の最良値が記されてある. 各近傍に対して, #best には決められた制限時間で探索を打ち切ったときに得られている暫定値が既知の最良値と一致した回数を, avr. cost には得られた 10 回の暫定値の平均値を記している. 探索の打ち切り時間は, 各近傍とも同じ値に設定した. 各問題例に対し, 3 つのアルゴリズムのうちで平均値が最小のものには*を付けている.

表1. 実験に用いた問題例に関する詳細

問題例	m	n	density	cost range
type e	1000	10000	2%	[1, 100]
type f	1000	10000	5%	[1, 100]
type g	1000	10000	2%	[1, 100]
type h	1000	10000	5%	[1, 100]
rail507	507	63009	1.2%	[1, 2]
rail516	516	47311	1.3%	[1, 2]
rail582	582	55515	1.2%	[1, 2]
rail2536	2536	1081841	0.4%	[1, 2]
rail2586	2586	920683	0.4%	[1, 2]
rail4284	4284	1092610	0.2%	[1, 2]
rail4872	4872	968672	0.2%	[1, 2]

問題の規模があまり大きくないタイプ e と f の問題例に対しては, 近傍による差がほとんど見られない. しかし, タイプ g, h, および rail の問題例においては, NB_1に比べ, NB_2と NB_3では, 解の精度が大幅に改善されている. この差に比べると, NB_2と NB_3の精度の差は小さいといえる. しかし, rail の大規模な問題例に対しては, NB_3を用いることによって解の質が向上していることが観測できる.

6.2 他のアルゴリズムとの比較

表3に, 代表的な近似アルゴリズムの実験結果との比較を記す. JB は Jacobs と Brusco によるアニーリング法を用いたアルゴリズム [9], BC は Beasley と Chu による遺伝アルゴリズム [3], CNS は Ceria, Nobil, Sassano による, CFT は Caprara, Fischetti, Toth によるラグランジュ緩和をベースにしたアルゴリズムであり [5][6], 表の値はそれぞれのアルゴリズムによって得られた最良値である. 我々の実験結果は, our LS に記してある. 実験は, 近傍 NB_3を用いて, 各問題に対し 10 回ずつ行った. min は制限時間内に得られた暫定値の最小値, max は得られた暫定値の最大値, avr. は 10 回の暫定値の平均値である. 探索の打ち切り時間は, タイプ e~h は 180 秒, rail 507~582 は 600 秒, rail 2536~4872 は 18000 秒とした. 文献 [7] に基づいて他のアルゴリズムとの計算時間を比較した. JB は 4 種類のパラメータ設定で各問題例に対し 5 回ずつ実験を行っており, 1 回の計算時間は我々のアルゴリズムの約 1/60 倍である. BC は各問題例に対しパラメータを変えて 10 回の実験を行っており, 1 回の計算時間は我々のアルゴリズムの約 1.5 倍である. CNS と CFT は各問題例に対し 1 回のみの実験を行い, その計算時間は我々のアルゴリズムと比べて, タイプ e~h に対してはほぼ同等, rail507~582 に対しては約 1/30 倍, rail2536~4872 に対しては約 1/6 倍である. 既知の最良値と等しいものには*を付けている. 表中の — は, 実験結果が報告されていないという意味である.

タイプ e~h については, 我々のアルゴリズムと CFT が全ての問題例に対して既知の最良値を得ている. rail 507~582 については, CNS, CFT, 我々のアルゴリズムの全てが既知の最良値を得ている. ただし, 我々のアルゴリズムに許した計算時間は, 他の 2 つのアルゴリズムの約 30 倍である. rail 2536~4872 については, rail 2536 に対しては既知の最良値が得られているが, 他の 3 つの問題例に対しては, CNS や CFT に比べて解の精度が悪く, 計算時間に関しても他の 2 つのアルゴリズムの約 6 倍程度である.

以上のことから, 我々のアルゴリズムは, タイプ e~h に対しては他のアルゴリズムと比較しても非常に高性能であるが, rail 2536~4872 のような非常に大規模な問題例に対しては, CNS や CFT に比べてやや性能が劣ることが観測できる. これは, 探索空間が大きすぎて, 制限時間内に十分な探索ができないためだと考えられる. 対策としては, 問題サイズの縮小方法を改良して探索の対象となる集合数をさらに削減したり, 状況に応じて探索する近傍サイズを小さくしたりして, アルゴリズムの高速化を図る予定である.

表 2. 異なる近傍による解の精度比較

問題例	best known	NB_1 #best (/10)	NB_1 avr. cost	NB_2 #best (/10)	NB_2 avr. cost	NB_3 #best (/10)	NB_3 avr. cost
e1	29	10	*29.0	10	*29.0	10	*29.0
e2	30	10	*30.0	10	*30.0	10	*30.0
e3	27	10	*27.0	10	*27.0	10	*27.0
e4	28	10	*28.0	10	*28.0	10	*28.0
e5	28	10	*28.0	10	*28.0	10	*28.0
f1	14	10	*14.0	10	*14.0	10	*14.0
f2	15	10	*15.0	10	*15.0	10	*15.0
f3	14	10	*14.0	10	*14.0	10	*14.0
f4	14	10	*14.0	10	*14.0	10	*14.0
f5	13	0	14.0	10	*13.0	10	*13.0
g1	176	10	*176.0	10	*176.0	10	*176.0
g2	154	0	156.0	10	*154.0	10	*154.0
g3	166	0	167.0	10	*166.0	10	*166.0
g4	168	0	171.0	0	170.0	10	*168.0
g5	168	7	168.3	10	*168.0	10	*168.0
h1	63	0	64.0	10	*63.0	2	63.8
h2	63	3	63.7	10	*63.0	10	*63.0
h3	59	0	60.0	5	*59.5	4	59.6
h4	58	5	58.5	10	*58.0	10	*58.0
h5	55	10	55.0	10	*55.0	10	*55.0
rail 507	174	0	179.2	6	*174.4	5	174.5
rail 516	182	1	182.9	10	*182.0	10	*182.0
rail 582	211	0	218.6	3	211.7	10	*211.0
rail 2536	691	0	712.4	1	*692.6	0	693.6
rail 2586	947	0	1000.9	0	962.2	0	*959.2
rail 4284	1065	0	1129.8	0	1082.9	0	*1078.3
rail 4872	1534	0	1614.4	0	1555.8	0	*1548.1

表3. 他のアルゴリズムとの比較

問題例	best known	JB (SA)	BC (GA)	CNS (LH)	CFT (LH)	our LS (10 runs) min	our LS (10 runs) max	our LS (10 runs) avr.
e1	29	*29	*29	—	*29	*29	*29	29.0
e2	30	*30	*30	—	*30	*30	*30	30.0
e3	27	*27	*27	—	*27	*27	*27	27.0
e4	28	*28	*28	—	*28	*28	*28	28.0
e5	28	*28	*28	—	*28	*28	*28	28.0
f1	14	*14	*14	—	*14	*14	*14	14.0
f2	15	*15	*15	—	*15	*15	*15	15.0
f3	14	*14	*14	—	*14	*14	*14	14.0
f4	14	*14	*14	—	*14	*14	*14	14.0
f5	13	14	*13	—	*13	*13	*13	13.0
g1	176	178	*176	*176	*176	*176	*176	176.0
g2	154	158	155	155	*154	*154	*154	154.0
g3	166	169	*166	167	*166	*166	*166	166.0
g4	168	172	*168	170	*168	*168	*168	168.0
g5	168	*168	*168	169	*168	*168	*168	168.0
h1	63	64	64	64	*63	*63	64	63.8
h2	63	64	64	64	*63	*63	*63	63.0
h3	59	60	*59	60	*59	*59	60	59.6
h4	58	59	*58	59	*58	*58	*58	58.0
h5	55	*55	*55	*55	*55	*55	*55	55.0
rail 507	174	—	—	*174	*174	*174	175	174.5
rail 516	182	—	—	*182	*182	*182	182	182.0
rail 582	211	—	—	*211	*211	*211	211	211.0
rail 2536	691	—	—	692	*691	*691	693	691.8
rail 2586	947	—	—	951	*947	954	958	955.6
rail 4284	1065	—	—	1070	*1065	1073	1078	1075.3
rail 4872	1534	—	—	*1534	*1534	1540	1545	1542.3

JB: simulated annealing by Jacobs & Brusco [9]

BC: genetic algorithm by Beasley & Chu [3]

CNS: Lagrangian-based heuristic by Ceria, Nobili & Sassano [6]

CFT: Lagrangian-based heuristic by Caprara, Fischetti & Toth [5]

7 まとめ

集合被覆問題に対し, より大きな近傍を探索する局所探索法に基づく近似解法を提案し, 計算実験を行った. 実験結果より, 我々のアルゴリズムは大きな近傍を高速に探索することができ, 大規模な問題に対しても適用できることが確認できた. 本研究で提案したアルゴリズムは他のアルゴリズムと比べて, ランダムに生成された問題に対しては良質の解を同程度の計算時間で求めることができるということが確認された. 今後は, 問題縮小の方法を工夫したり, 状況に応じて探索する近傍のサイズを変化させるなどして, アルゴリズムの改善を図る予定である.

参考文献

[1] E.K. Baker, L.D. Bodin, W.F. Finnegan and R.J. Ponder, "Eficient Heuristic Solutions to an Airline Crew Scheduling Problem," *A.I.I.E. Trans*, 11 (1976) 79-85.

[2] J.E. Beasley, "A Lagrangean Heuristic for Set Covering Problems," *Naval Research Logististics*, 37 (1990) 151-164.

[3] J.E. Beasley and P.C. Chu, "A Genetic Algorithm for Set Covering Problem," *European Journal of Operational Research*, 94 (1996) 392-404.

[4] E. Boros, P. L. Hammer, T. Ibaraki and A. Kogan, "Logical Analysis of Numerical Data," *Mathematical Programming*, 79 (1997) 163-190.

[5] A. Caprara , M. Fischetti and P. Toth, "A Heuristic Method for the Set Covering Problem," *Proceedings of the Fifth IPCO Conference*, Springer-Verlag, (1996) 72-81.

[6] S. Ceria , P. Nobili and A. Sassano, "A Lagrangian-based heuristic for large-scale set covering problems," *Mathematical Programing*, 81 (1998) 215-228.

[7] J.J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software," Technical Report No. CS-89-85, Computer Science Department, University of Tennessee, July 1998.

[8] M.L. Fisher and P. Kedia, "Optimal Solutions of Set Covering/Partitioning Problems Using Dual Heuristics," *Management Science*, 36 (1990) 674-688.

[9] L.W. Jacobs and M.J. Brusco, "A Local-Search Heuristic for Large Set-Covering Problems," *Naval Research Logististics*, 42 (1995) 1129-1140.

[10] B.M. Smith, "IMPACS–A Bus Crew Scheduling System Using Integer Programing," *Mathmatical Programing*, 42 (1988) 181-187.

[11] F.J. Vasko and G.R. Wilson, "Using a Facility Location Algorithm to Solve Large Set Covering Problems," *Operations Research Letters*, 3 (1984) 85-90.