アルゴリズム実装を教える

梅谷　俊治

学生がアルゴリズムやプログラミングを授業で教わるだけではなく，実際に自らの手を動かしてアルゴリズムを実装することは，アルゴリズムに対する理解を深め，問題解決のプロセスを体験する良い機会でもある．本稿では，筆者が大学 3 年生を対象に実施した組合せ最適化問題に対するアルゴリズム実装の実習について，その概要と具体的な課題内容を紹介する．

キーワード：アルゴリズム，組合せ最適化，巡回セールスマン問題，長方形詰込み問題，グラフ彩色問題

1. はじめに

学生がアルゴリズムやプログラミングの授業を受けても，すぐにアルゴリズムを実装できるわけではない．多くの学生は，いざプログラムを書き始めると，アルゴリズムに対する理解が不十分であったり，教科書に記載されたアルゴリズムと実際のプログラムの間に大きな隔たりがあることに気づかされる．学生が自らの手を動かしてアルゴリズムを実装することは，アルゴリズムに対する理解を深め，問題解決のプロセスを体験する良い機会である．しかし，アルゴリズムの実装に関する体系的なカリキュラムがあるわけではなく，高等教育の現場では個々の教員が個人の経験に基づいて試行錯誤を繰り返しながら学生を指導しているのが現状である．本稿では，筆者が大学 3 年生を対象に実施した組合せ最適化問題に対するアルゴリズム実装の実習について，その概要と具体的な課題内容を紹介する．この実習で用いたガイダンス資料，問題例，プログラムは筆者のウェブサイト[1]にて公開しているので，興味を持たれた方はそちらも参照いただきたい．

2. アルゴリズム実装の実習概要

筆者は，2005 年度から 2007 年度まで電気通信大学電気通信学部システム工学科においてシステム工学実験の一部を担当した．このシステム工学実験は複数の課題で構成されるオムニバス形式の実習科目で，1 課題当たり 3 週間の期間で合計 9 コマ（13.5 時間）の実習時間が割り当てられている．組合せ最適化問題に対するアルゴリズム実装では，始めの 1.5 時間を課題内容の説明を含むガイダンスに，残りの 12 時間を実習に割り当てた．

この実習の対象は 3 年生で，アルゴリズムとプログラミングの授業を受けたもののアルゴリズムを実装した経験がない学生が大半である．また，この実習では，学生は C 言語の文法を知っていることを前提にしているが，これは当該学科の学生が前年度に C 言語の授業を受けたことが理由で，手続き型の言語であれば C 言語でなくても構わない．

この実習では以下の点に気をつけて問題とアルゴリズムを決定した．

1. **標準的な教科書に掲載されている問題やアルゴリズムを使わない．**
   標準的な教科書に掲載されている問題やアルゴリズムは書籍やウェブサイトでプログラムが公開されている場合が多いため，この実習ではやや発展的な問題である巡回セールスマン問題（2005 年度），長方形詰込み問題（2006 年度），グラフ彩色問題（2007 年度）を課題として取り上げた．
2. **プログラムの実行結果を図で確認できる．**
   いずれの課題でも，学生がプログラムの実行結果を図で確認できるように，出力結果を図示するビューワを用意した．また，学生が効率良くデバッグ作業を進められるように，出力解が制約条件を違反している場合はその部分をハイライトする機能を付けた．当初は予想していなかったが，実行結果が目で確認できるとモノを作っている実感が湧くようで，学生のやる気を引き出すのに一役買っていた．

うめたに　しゅんじ
大阪大学大学院情報科学研究科
〒 565–0871 大阪府吹田市山田丘 2–1

[1] http://www-sys.ist.osaka-u.ac.jp/~umetani/software_ja.html

3. **高度なデータ構造や再帰呼び出しを用いなくてもアルゴリズムを実装できる.**
   アルゴリズムを実装した経験がない学生が多いことを考慮して，高度なデータ構造や再帰呼び出しを使わなくても実装できるアルゴリズムを課題に取り上げた.
4. **プログラムがデータ入出力の手続きを除いて100行で収まる.**
   プログラミングに使える時間が少ないことを考慮して，データ入出力の手続きを書いたサンプルプログラムを用意した．また，プログラムを書く量をできる限り減らす一方で，アルゴリズムの詳細な手続きや小さな問題例に対する実行過程を紙に書き出して確認する作業に時間を割くように学生に指示を出した.「モニタとにらめっこしてもアルゴリズムは正しく動くようにはならない」と学生に声を掛けて回ったことは今でもよく覚えている.

3. 巡回セールスマン問題

n 個の都市と都市 i と都市 j の距離 d_{ij} が与えられたときに，すべての都市をちょうど一度ずつ訪れて最初の都市に戻る巡回路のうち，総移動距離が最小となるものを求める問題が巡回セールスマン問題である．この実習では，平面上に n 個の都市とその位置を与えて，距離 d_{ij} を都市 i と都市 j のユークリッド距離とした．すなわち，都市 i と都市 j の座標を (x_i, y_i)，(x_j, y_j) とするとき，$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ と定義する.

学生には，この巡回セールスマン問題に対して，最近近傍法 (nearest neighbor algorithm)，最近追加法 (nearest addition algorithm)，2-opt 法 (2-opt algorithm) の 3 つの近似解法のプログラムを作成する課題を与えた．また，プログラムの動作確認のために，TSPLIB[2]から取得したベンチマーク問題例（51〜2,392 都市の 16 問）を与えた.

近似解法は，何もないところから 1 つずつ都市を追加して巡回路を作る構築法と，すでにある巡回路に変形を加える操作を繰り返し適用してより短い巡回路を作る改善法に分類され，最近近傍法と最近追加法が構築法，2-opt 法が改善法に該当する.

最近近傍法は，適当な都市から出発して，現在の都市から最も近い未訪問の都市に移動する操作をすべての都市を訪問するまで繰り返した後に出発した都市に戻る解法である．最近近傍法は単純で学生が手始めに取り組むのに向いているが，最後に訪問する都市が出発した都市から遠く離れてしまう場合が少なくない.

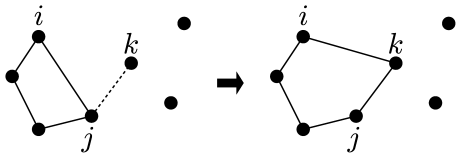

図 1　最近追加法の実行例

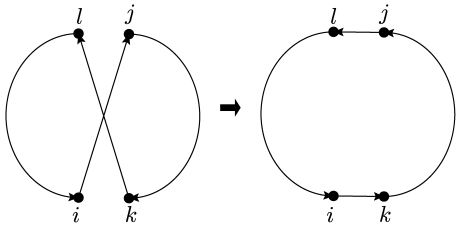

図 2　2-opt 法の実行例

最近追加法は，適当な都市を 1 つ含む部分巡回路から始めて，部分巡回路に都市を 1 つずつ追加する操作をすべての都市が部分巡回路に含まれるまで繰り返す解法である．図 1 に示すように，各反復では，部分巡回路に含まれる都市 j と含まれない都市 k のすべての組合せの中で d_{jk} を最小にする組を求めた後に，都市 j の直前（もしくは直後）に都市 k を訪問することで部分巡回路を更新する．ちなみに，最近追加法は最適な巡回路の 2 倍以内の長さの巡回路を常に出力する精度保証付き近似解法である.

2-opt 法は，与えられた巡回路から 2 本の辺を取り除いた後に，再び巡回路となるように別の 2 本の辺を加えて繋ぎ直し，より短い巡回路が得られたら元の巡回路と置き換える手続きを繰り返す解法である．図 2 に示すように，取り除く辺を (i, j)，(k, l) とすると加える辺は (i, k)，(j, l) と 1 通りに定まるため，$d_{ik}+d_{jl} < d_{ij}+d_{kl}$ が成立する場合のみ巡回路を更新すれば良いことがわかる.

アルゴリズムを実装する際には，解をどのようなデータ構造で格納して各操作を実現するかよく考える必要がある．この実習では，出力結果を図示するビューワを利用するため，n 要素の配列 x を用いて $x[0], x[1], \ldots, x[n-1]$ に訪問順序に従って都市の番号を格納して解を表す．しかし，最近追加法では，部分

[2] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

巡回路に都市を 1 つずつ追加する操作を繰り返すため，アルゴリズムの内部では，n 要素の配列 p を用いて $p[j]$ に都市 j の直前に訪問する都市の番号を格納したほうが効率は良い．図 1 の例では，初めは都市 i, j の順に訪問するため $p[j] = i$ となるが，都市 k を追加すると $p[k] = i$，$p[j] = k$ と更新される．2-opt 法では，配列 $x = (\dots, i, j, \dots, k, l, \dots)$ の j から k の部分列を反転させることで，辺の入替えに対応する操作を実現する．また，2-opt 法ではランダムな順列を初期解として与えるが，これを生成する方法も学生には自明ではないため，ランダムな順番に都市を訪問する巡回路を出力するサンプルプログラムを学生に配布した．

ランダムな順列の生成

Step 1: $j \leftarrow n-1$ とする．

Step 2: 区間 $(0,1)$ に一様に分布するランダムな値 U を生成し，$k \leftarrow \lfloor jU \rfloor$ とする．$x[j]$ と $x[k]$ の値を入れ替える．

Step 3: j を 1 減らし，$j > 0$ ならば Step 2 へ戻る．そうでなければ終了する．

いずれの近似解法も短いプログラムで実現できるがうまく工夫すると計算の効率を改善できる．例えば，最近追加法では，各反復で部分巡回路に含まれる都市 j と含まれない都市 k のすべての組合せの中で d_{jk} を最小にする組を求めるため，何の工夫もない単純な実装だとアルゴリズムの実行時間は $\mathrm{O}(n^3)$ となる．しかし，部分巡回路に含まれる各都市 j に対して，部分巡回路に含まれない最も近い都市の番号とその距離を補助情報として配列に記憶しておき，各反復で部分巡回路に新たに追加された都市 k に関わる部分のみ補助情報を更新すれば，その分だけプログラムの処理は複雑になるものの実行時間を $\mathrm{O}(n^2)$ に減らすことができる．もちろん，まず正しく動作するプログラムを完成することを最優先すべきなので，初めのガイダンスでは計算の効率には全く触れずにおいて，進度が早く手持ち無沙汰になりそうな学生にのみヒントを与えてアルゴリズムの改良に挑戦するように促した．

巡回セールスマン問題に対する近似解法とその実装については [1〜3] が詳しい．また，Cook のウェブサイト[3]には巡回セールスマン問題の問題例，プログラムを含む多くの資料が公開されている．

4. 長方形詰込み問題

図 3 に示すように，幅が固定で十分な高さがある長方形の容器と n 個の長方形の荷物が与えられる．容器の幅を W，各荷物 i の幅を w_i，高さを h_i とする．荷物はその下辺が容器の下辺と平行になるように配置し，回転は許さないものとする．ここで，すべての荷物を互いに重ならないように配置する．(x_i, y_i) を荷物 i の左下隅の座標を表す変数とすると（容器の左下隅を原点とする），問題の制約条件は以下のとおりに記述できる．

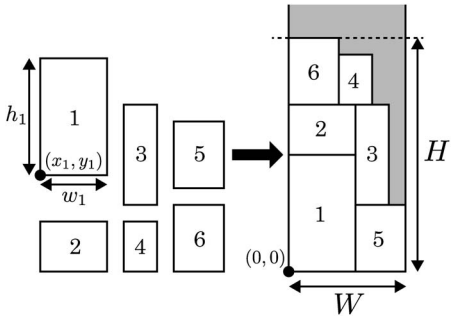

図 3 長方形詰込み問題の例

制約条件 1：荷物 i は容器内に配置される．これは，以下の 2 本の不等式がともに成り立つことと同値である．

$$\begin{aligned} 0 \le x_i \le W - w_i, \\ 0 \le y_i \le H - h_i. \end{aligned} \tag{1}$$

制約条件 2：荷物 i と荷物 j は互いに重ならない．これは，以下の 4 本の不等式のうち 1 本以上が成り立つことと同値であり，各不等式はそれぞれ荷物 i が荷物 j の左側，右側，下側，上側にあることを記述している．

$$\begin{aligned} x_i + w_i \le x_j, \\ x_j + w_j \le x_i, \\ y_i + h_i \le y_j, \\ y_j + h_j \le y_i. \end{aligned} \tag{2}$$

このとき，制約条件を満たしたうえで，必要な容器の高さ H を最小にする荷物の配置を求める問題が長方形詰込み問題[4]である．

学生には，この長方形詰込み問題に対して，NF 法 (next-fit algorithm)，NFDH 法 (next-fit decreasing height algorithm)，BLF 法 (bottom-left-fill algorithm) の 3 つの近似解法のプログラムを作成する課題を与えた．また，プログラムの動作確認のために ESICUP[5]から取得したベンチマーク問題例（長方形が 10〜3,152 個の 23 問）を与えた．

NF 法および NFDH 法は，ビンパッキング問題に対

[3] http://www.math.uwaterloo.ca/tsp/index.html

[4] 正確には長方形ストリップパッキング問題と呼ばれる．

[5] http://paginas.fe.up.pt/~esicup/

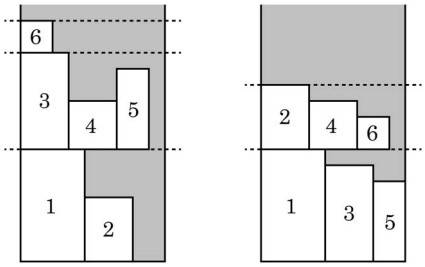

図 4　NF 法（左），NFDH 法（右）の実行例

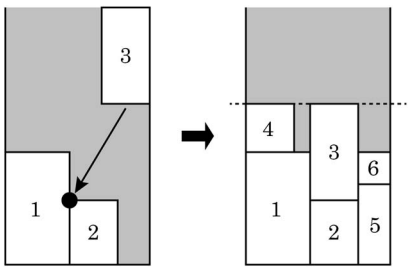

図 5　BLF 法の実行例

する近似解法を長方形詰込み問題に合わせて拡張した解法であり，NF 法は番号順に，NFDH 法は高さの降順に長方形を 1 つずつ配置する．図 4 に示すように，いずれの解法も容器を水平な直線でレベルと呼ばれる領域に分割して長方形を詰込むためレベル法と呼ばれる．左端に配置された長方形が各レベルの高さを決定しており，長方形を左端から順に横一列に配置し，現在のレベルの中に配置できない長方形が現れたら新たなレベルを生成してその長方形を左端に配置する．

NF 法のアルゴリズムの実行時間は $O(n)$，NFDH 法のアルゴリズムの実行時間は長方形を高さの降順に並び替えるため $O(n \log n)$ となる．また，NFDH 法は最適解の高さの 3 倍以内の解を常に出力する精度保証付き近似解法である．この他にも類似の解法として，長方形を常にできる限り下のレベルに配置する FF 法 (first-fit algorithm) や，長方形を常にできる限り隙間の小さいレベルに配置する BF 法 (best-fit algorithm) などが知られている．

BLF 法は，与えられた順に従って長方形を 1 つずつできる限り左下隅に配置する解法である．長方形を下にも左にも動かせないような配置は BL 安定点と呼ばれ，図 5 に示すように，BLF 法は最も下で同じ高さであれば最も左にある BL 安定点に長方形を配置する．

図 6 に示すように，BL 安定点は，(i) 容器の左下隅，(ii) 長方形 i の右辺を下に伸ばした半直線と容器の下辺の交点，(iii) 長方形 i の上辺を左に伸ばした半直線と容器の左辺の交点，(iv) 長方形 i の右辺を下に伸ばした半直線と長方形 j の上辺を左に伸ばした半直線の交点のみを候補として列挙すれば良い．しかし，配置する長方形の形状によっては左もしくは下に動かせる場合や他の長方形と重なりが生じる場合があるため，各候補に対してこれらの条件を確認する必要がある．

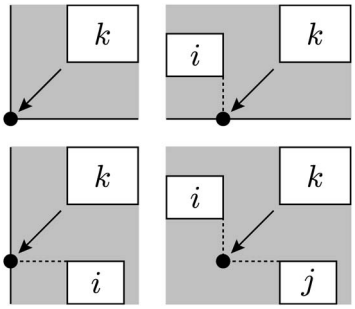

図 6　BL 安定点の例

BLF 法では，各反復で BL 安定点の各候補に対して長方形が配置可能かどうか確認するため，単純な実装だとアルゴリズムの実行時間は $O(n^3)$ となる．ただし，BL 安定点の候補を補助情報として配列に記憶しておき，各反復で新たに配置された長方形 k に関わる部分のみ補助情報を更新すれば，その分だけプログラムの処理は複雑になるものの実行時間を大幅に減らすことができる．また，BLF 法では，長方形の配置順が解の精度に大きく影響を与えるため，面積の降順に長方形を配置するアルゴリズムや局所探索法 [2, 4, 5] を用いてよい配置順を探索するアルゴリズムの実装を発展的な課題として追加した．

この実習では，NF 法，NFDH 法，BLF 法の 3 つを課題として与える予定であった．しかし，BLF 法は各反復で BL 安定点を探索する手続きが複雑で，多くの学生にとって実装は容易ではないことが準備段階でわかったため，NF 法と NFDH 法を完成できれば合格とした．

長方形詰込み問題に対する近似解法については [6, 7] が詳しい．

5.　グラフ彩色問題

n 個の頂点と m 本の辺を持つ無向グラフが与えられたとき，辺 (i, j) の両端の頂点 i と頂点 j が同色にならないようにすべての頂点を彩色する．このとき，必要な色数を最小にする頂点の彩色を求める問題がグラフ彩色問題である．

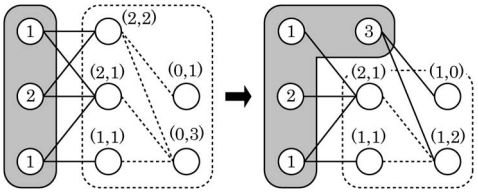

図 7　DSATUR 法の実行例

学生には，このグラフ彩色問題に対して，SEQ 法 (sequential coloring algorithm)，DSATUR 法 (degree of saturation algorithm)，RLF 法 (recursive largest first algorithm) の 3 つの近似解法のプログラムを作成する課題を与えた．また，プログラムの動作確認のために，ランダム幾何グラフ（10〜1,000 頂点の 18 問）を与えた．ランダム幾何グラフは頂点数 n，パラメータ d を入力とし，平面上の単位正方形 $[0,1]^2$ 内に一様に n 個の頂点を分布させ，距離が d 以下の 2 頂点間を辺で結び生成されるグラフである．

SEQ 法は，番号順に頂点を 1 つずつ彩色する方法で，各反復では，頂点 i に対して彩色可能な（隣接する頂点で使われていない）最小の色番号を割り当てる．頂点の最大次数（隣接する頂点の数）を Δ とすると，SEQ 法は，$\Delta+1$ 色以下の解を常に出力するが，次数の降順に頂点を 1 つずつ彩色することで，より色数の少ない解を求められることが経験的に知られている．

DSATUR 法は，隣接する頂点で使われている色数を飽和次数，隣接する未彩色の頂点の数を残余次数と定義し，飽和次数が最大，同点の場合は残余次数が最大となる頂点 i を選び，彩色可能な最小の色番号を割り当てる操作をすべての頂点が彩色されるまで繰り返す解法である．図 7 は DSATUR 法の実行例で，頂点内の数字が割り当てられた色番号を，頂点近くの数字の組が飽和次数と残余次数を表す．SEQ 法，DSATUR 法は単純な実装だとアルゴリズムの実行時間は $O(n^2)$ となる．

RLF 法は，同じ色番号を割り当てられる（互いに隣接していない）頂点の集合を安定集合と定義し，極大な安定集合を見つけて，それに含まれるすべての頂点に新しい色番号を割り当てる操作をすべての頂点が彩色されるまで繰り返す解法である．各反復では，まず残余次数が最大となる未彩色の頂点を選び新しい色番号 k を割り当てる．色番号 k を割り当てた頂点と隣接する未彩色の頂点は色番号 k を割り当てられないため集合 U にまとめる．以降は，図 8 に示すように，集合 U に含まれない未彩色の頂点で集合 U に含まれる

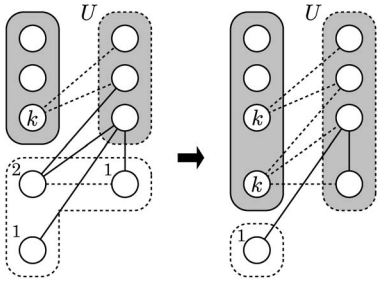

図 8　RLF 法の実行例

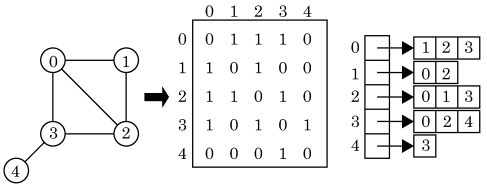

図 9　グラフの隣接行列（左）と隣接リスト（右）

頂点に張る辺の数が最大となる頂点を選び色番号 k を割り当てる操作を繰り返す．RLF 法は単純な実装だとアルゴリズムの実行時間は $O(n^3)$ となる．

グラフのデータ構造はアルゴリズムの計算の効率に大きく影響を与える．隣接行列はグラフを表現する最も単純なデータ構造で，頂点 i と頂点 j を繋ぐ辺があれば行列の (i,j) 要素と (j,i) 要素をともに 1 とする．隣接リストはグラフを表現するのに一般的に使われるデータ構造で，各頂点に対して隣接するすべての頂点を配列やリストに持つ．図 9 にグラフの隣接行列と隣接リストの例を示す．隣接行列は $n \times n$ の 2 次元配列を 1 つ用意すれば済む一方で，隣接リストは各頂点に対してその次数と同じ長さの配列やリストを用意する必要があり，データ構造が複雑な分だけプログラムの処理も複雑となる．しかし，隣接リストを用いて実装すれば，SEQ 法と DSATUR 法のアルゴリズムの実行時間を $O(n\Delta)$ に，RLF 法のアルゴリズムの実行時間を $O(m\Delta)$ に減らすことができる．もちろん，まず正しく動作するプログラムを完成することを最優先すべきなので，初めに配布するサンプルプログラムではグラフを隣接行列に格納し，進度が早く手持ち無沙汰になりそうな学生にのみヒントを与えて隣接リストを用いた実装に挑戦するように促した．

グラフ彩色問題に対する近似解法とその実装については [2, 8] が詳しい．

6. 実習の取り組みと問題点

学生の大半はこれまでアルゴリズムを実装した経験がないため，プログラムを書き始める前に小さな問題例を作ってアルゴリズムの実行過程を紙の上で図を交えて再現するように指示した．これは，アルゴリズムの仕組みの理解するためだけではなく，プログラムの動作確認やデバッグにも利用する．

プログラムを一度にすべて書いてから動作確認をするとバグが生じた箇所とその原因を特定することが難しくなるため，プログラムを 1 ブロック書いては動作確認してバグがないことを確かめながら少しずつ実装を進めるように指示した．また，アルゴリズムを実装した経験がない学生は，コンパイル時のエラーがなくなれば正しいプログラムが書けたと勘違いしがちなので注意する必要がある．

多くのバグはそれが生じた箇所から離れた箇所に原因が潜んでいる場合が多い．例えば，変数値の初期化を忘れる，参照する配列の要素を間違えるなどの誤りが原因となり，そこから離れた箇所でバグを引き起こすことは少なくない．そこで，典型的なバグがどのような原因で生じるのかを理解してもらうために，printf 関数[6]を挿入して配列や変数の値が意図したとおりになっているかこまめに確認するように指示した．

同一のファイルでプログラムの修正を繰り返すと，正しく動作していたプログラムが動かなくなった場合に収拾がつかなくなるので，こまめにプログラムのバックアップを取るように指示した．もちろん，将来はデバッガやバージョン管理システムを利用するべきであるが，この実習では学生にデバッグとバージョン管理の重要性を認識させる程度にとどめた．

アルゴリズムを実装する際には，プログラムの動作確認や性能評価のため，計算結果の再現性と出力解の正当性を保証することが必要である．例えば，乱数の種に実行時の時刻を用いることは原則として避けるべきである．余談であるが，C 言語では乱数の種を設定せずに rand 関数を呼び出すと乱数の種は 1 に設定されるが，Python では乱数の種を設定せずに random 関数を呼び出すと乱数の種は実行時の時刻に設定されてしまうので要注意である．また，プログラムが正しく動作していることを確認するために，プログラムの最後に出力解に対して制約条件の充足と目的関数の値を再確認する手続きを書くべきである．

プログラミングの経験がない学生に対して，一度に多くの注意を与えると混乱してしまうのでタイミングを見計らって少しずつ注意するよう気をつける必要がある．一方で，自分の書いたプログラムに間違いがあるはずがないと高をくくる学生も少なくないので，周りくどいと感じてもバグがないことを確認しながら少しずつ実装を進めることの重要性を繰り返し説く必要がある．

この実習では，適当なタイミングを見計らって筆者自身が作成したプログラムのデモを実施した．これは，ほんの少し工夫で計算の効率が劇的に改善できることを学生に知らしめるためで，実際に計算時間が数十〜数百分の一に短縮されるのを目の当たりにした学生はアルゴリズムの奥深さを実感できたのではないかと思う．

最後に，プログラミング実習における問題点について少し触れる．この実習で与えた課題はアルゴリズムを実装した経験のない学生にはやや難易度が高く，独りで実習を進めることは容易ではなかったため，学生同士で互いに相談しながら実習を進めるよう指示した．その結果，学生が作成したプログラムが似通ったものになり，丸写ししたものとほとんど区別がつかなくなる状況が少なからず生じたが，最後まで有効な対応策を考えつくことはできなかった．

7. おわりに

本稿では，筆者が大学 3 年生を対象に実施した組合せ最適化問題に対するアルゴリズム実装の実習について，その概要と課題内容を紹介した．本稿で紹介したノウハウの多くはアルゴリズムを実装する経験を積めば自然と身につくものだが，限られた実習時間の中でほとんど経験がない学生にこれらのノウハウを教え，学生が自分自身の力でプログラムを完成できるように導くためには周到な準備と多くの工夫が必要となる．

この実習のカリキュラムは，筆者自身の経験に基づく試行錯誤の結果で最良のカリキュラムには遠く及ばないが，アルゴリズム実装の指導に関わる方々に少しでも参考になれば幸いである．

[6] printf 関数の内容を直ちに標準出力に反映するためには fflush 関数を直後に追加する必要がある．

参考文献

[1] D. S. Johnson and L. A. McGeoch, “The traveling salesman problem: A case study,” *Local Search in Combinatorial Optimization*, E. H. L. Aarts and J. K. Lenstra (eds.), John Wiley & Sons, pp. 215–310, 1997.

[2] 久保幹雄，J. P. ペドロソ，『メタヒューリスティクスの数理』，共立出版，2009.

[3] 山本芳嗣，久保幹雄，『巡回セールスマン問題への招待』，朝倉書店，1997.
[4] 梅谷俊治，柳浦睦憲，“メタヒューリスティクス事始め—まずは局所探索法から—，”オペレーションズ・リサーチ，**58**, 689–694, 2013.
[5] 柳浦睦憲，茨木俊秀，『組合せ最適化—メタ戦略を中心として—』，朝倉書店，2001.
[6] S. Imahori, M. Yagiura and H. Nagamochi, “Practical algorithms for two-dimensional packing,” *Handbook of Approximation Algorithms and Metaheuristics*, T. F. Gonzalez (ed.), Chapman & Hall/CRC, 36/1–15, 2007.
[7] 今堀慎治，梅谷俊治，“切出し・詰込み問題とその応用— (2) 長方形詰込み問題—，”オペレーションズ・リサーチ，**50**, 335–340, 2005.
[8] D. S. Johnson, C. R. Aragon, L. A. McGeoch and C. Schevon, “Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning,” *Operations Research*, **39**, 378–406, 1991.