離散凸解析ソルバODICONとWebアプリケーション

土村　展之

離散凸解析理論の成果である各種のアルゴリズムを，C 言語プログラムの形で提供するのが ODICON である．ODICON は Optimization algorithms for DIscrete CONvex functions からとったものであり，「オダイコン」と読む．アプリケーションソフトウェアとデモンストレーションソフトウェアを開発して Web 上に公開した．アルゴリズムの詳細を理解しなくても離散凸解析関連の研究や応用事例研究が行えるような環境を整備することを目指した取り組みである．

キーワード：離散凸関数最小化，劣モジュラ関数，最適化ソルバ，Web デモ

1. ソフトウェア開発の動機

半正定値計画問題に対しては，本誌の 2010 年 7 月号の特集記事にもあるとおり，小島政和氏のグループによって SDPA という高性能なソルバが開発されている．このソフトウェアの存在が，世の中で半正定値計画問題に定式化して解いてみようと思う人を発掘する役目も果たしていて，そのおかげで応用例も明らかになっていることであろう．理論と実用の良い循環があるように見える．

われわれもこれをモデルに開発を始めた．つまり，離散凸解析における最小化問題のソルバを開発して公開し，世の中にある問題を効率的に解くことに貢献し，実際的な問題例も手に入れたいと考えた．理論上解けることと，実際の問題がどのように解けるか（解けないか）にはおそらくギャップがあり，そのギャップに直面すれば，理論的な研究にも新たな課題が見えてくるであろう．また，現在の世の中では，L 凸性／M 凸性に気づかないまま最適化が行われている関数も多くあると想像する．そのようなものの L 凸性／M 凸性が明らかになれば，汎用エンジンで高速に解ける可能性も出てくる．このように，離散凸解析の理論面と実用面の双方にとって，ソフトウェア開発は意義のあることだと考えて取り組みを始めた．

ODICON（オダイコン）[9] の名前の由来を紹介する．筆者はすでにソフトウェアプロジェクトとして，TEX にまつわる `ptetex3` や `ptexlive` といったものを手がけていたが，無機質な名称のため，満足していなかった．

このような状況で新たなソフトウェア開発を始めるにあたり，自分で名称を考えることに限界を感じた筆者は，共同研究者の M 女史に「ソフトウェアの名前は重要なんですよね，普及するかどうかの鍵を握ってて．いい名前ないですかね，適度に母音が混ざってると読みやすいかも．」とリクエストしてみた．そうして提案してもらったいくつかの名称の中から，語呂の良さと，「お大根」の可愛らしさから ODICON (Optimization algorithms for DIscrete CONvex functions) を採用するに至った．しかしあとから判明したことに，関東では大根に「お」は付けないらしい．どうやら関西出身の 2 人の言語文化が発露してしまったようである．

2. 離散凸関数最小化アルゴリズム

$L^\natural$ 凸関数や $M^\natural$ 凸関数の最小化のためのアルゴリズムには，いろいろなものが提案されている．ここでは，特に ODICON に関係の深いものについて簡単に述べる．なお，$L^\natural$ 凸関数と $M^\natural$ 凸関数について紹介するが，L 凸関数と M 凸関数についても，それぞれ $\natural$（ナチュラル）付きのものと等価に変換ができるので，アルゴリズムの存在や計算量についての状況は同じである．

2.1 最急降下法

離散関数最小化における最急降下法（貪欲アルゴリ

つちむら　のぶゆき
関西学院大学理工学部
〒 669–1337 兵庫県三田市学園 2 丁目 1 番地

ズムなどとも呼ばれる）は，誤解を恐れずに言うと，組合せ最適化問題におけるローカルサーチ（近傍探索法）とほぼ同じアプローチである．暫定解の周辺の近傍を探索し，目的関数がもっとも改良されるところに解の更新を行う，という点については同じ動作をする．一般の組合せ最適化問題と異なるのは，離散凸関数はうまくできていて，離散凸関数の種類に応じて近傍が厳密に定められており，その近傍を探索するだけで最終的に大域最適解（最小解）にたどりつくことが保証されていることである．

$L^\natural$ 凸関数における近傍は，次の定理 2.1 をそのまま利用すると，$O(2^n)$ と指数個になる [1]．

定理 2.1 関数 $g : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ が $L^\natural$ 凸関数のとき，$g(p) < +\infty$ を満たす整数ベクトル p が g の最小点であるためには，任意の $q \in \{0,1\}^n$ に対して

$$g(p) \le \min\{g(p-q),\ g(p+q)\}$$

となることが必要十分である．

この近傍をすべて探索するのは現実的ではないが，劣モジュラ集合関数最小化アルゴリズムを適用すると，すべての近傍を探索する必要がなくなり，大幅な高速化が可能となる．ODICON では，Iwata–Fleischer–Fujishige [3] の組合せ論的な多項式時間アルゴリズム（IFF と略記）と最小ノルム基底を利用する Fujishige–Wolfe アルゴリズム（FW と略記）[2] を用いている（第 3 節参照）．

$M^\natural$ 凸関数における近傍は，次の定理 2.2 のように小さく，定義どおりに探索を行っても $O(n^2)$ 個ですむ．以下では第 i 単位ベクトルを $e_i (\in \{0,1\}^n)$ と表す．ただし e_0 だけは特別で，$e_0 = (0,0,\ldots,0)$ とする．

定理 2.2 関数 $f : \mathbb{Z}^n \to \mathbb{R} \cup \{+\infty\}$ が $M^\natural$ 凸関数のとき，$f(x) < +\infty$ を満たす整数ベクトル x が f の最小点であるためには，任意の $i, j \in \{0, 1, \ldots, n\}$ に対して

$$f(x) \le f(x - e_i + e_j)$$

となることが必要十分である．

さらに，近傍の探索順序の順序を工夫するなどの高速化が可能であり，提案されている最急降下法には，

(i) $f(x - e_i + e_j)$ を最小化する (i, j) を見いだす基本形（[1, 227 頁] の「降下法」），

(ii) 適当に選んだ i に対して $f(x - e_i + e_j)$ を最小化する j を見いだす修正形（[5] の ‘Modified_Steepest_Descent’），

(iii) 修正形 (ii) に領域縮小を組み込んだもの（[8, 306 頁] の ‘Greedy’）

のようなバリエーションがある．

なお，$L^\natural$ 凸関数と $M^\natural$ 凸関数のどちらの場合でも，1 反復の近傍探索は多項式時間で完了できるが，最小解にたどりつくまでに必要な反復数が，初期解からの距離に比例するため，全体としては多項式時間アルゴリズムにはならない[1]．

2.2 スケーリング法

前出の最急降下法が，大域最小解までの距離が遠くても近くても，一歩ずつ着実に進むのに対して，スケーリング法は，最初は歩幅を大きくして進み，だんだんと歩幅を縮めていき，最後には最急降下法と同じ歩幅で最小解にたどりつくというものである．

当然ながら，実用的にもスケーリング法のほうが高速であり，理論的にもこの手法を使って多項式時間アルゴリズムを作ることができる．

なお，組合せ問題の文脈ではスケーリング法を説明することができない．なぜなら，組合せ問題の解空間には 0–1 変数や順列などが用いられ，歩幅に対応する概念（ある解からある方向に距離を直線的に伸ばす）が考えられないからである．

2.3 連続緩和法

ある関数の，連続変数のものと離散変数のものを思い浮かべたい．まず連続変数の関数 $\bar{f} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ が与えられるとする．次に定義域を整数ベクトルに制限して離散変数の関数として採用する．つまり，

$$f(x) = \bar{f}(x) \qquad (\forall x \in \mathbb{Z}^n) \tag{1}$$

として f を考える．

このとき，どちらが最小化しやすいかと問われれば，関数の持つ性質にもよるが，一般には連続変数のほうがはるかに解きやすい．このことは，多くの人の直感には反する事実であろう．なぜ連続版のほうがやさしいのかというと，微分なり勾配なり，使える道具がいろいろあるからである．関数値が下がりそうかどうかは，周囲の状況から推測しやすい．ところが離散版だと，飛び飛びの情報しかないので，緩やかに変化するということがない．関数値がどうなるかは，周囲の状況から推測するにも情報が乏しく，結局はその地点を直

[1] 擬多項式時間アルゴリズムである．

接調べたほうが手っ取り早いということが往々にしてある．こういった事情で離散版のほうが手間がかかる．

さて，連続変数が解きやすいのであれば，それを利用しようというのが連続緩和法である．離散 $L^\natural$ 凸/$M^\natural$ 凸関数 f とその連続版 $\bar{f}$ が手に入ったとき（$\bar{f}$ は自動的に凸関数になる），連続版の最小解を求めて，その近くの整数格子点から離散版の探索を始める．

離散解と連続解の距離は一般に近く，その距離には理論的な裏づけもある．仕上げの離散解の探索にはスケーリング法の必要はなく，むしろ単純な最急降下法で十分である．連続最小化が速いおかげで，全体としてもスケーリング法より高速であるが，連続版の $\bar{f}$ がうまく手に入るかどうかが，この手法が使えるかどうかの鍵となる．理論的には連続版の凸関数 $\bar{f}$ は存在することがわかっているが，ODICON では $\bar{f}$ を準ニュートン法で最小化しているので，連続かつ微分（差分近似）可能である必要があり，このような条件を満たしたものが作れるかどうかは明らかではない．

3. 離散凸関数最適化ソルバ

本節では，開発した ODICON の機能を紹介する [10]．実装した離散凸関数最小化アルゴリズムは，L / $L^\natural$ / M / $M^\natural$ 凸関数のそれぞれに対する最急降下法（第 2.1 節），スケーリング法（第 2.2 節），連続緩和法（第 2.3 節）である．L 凸関数と $L^\natural$ 凸関数は，理論的には等価であって，互いに変換可能ではあるが，利用者の利便性を考えて，どちらの最小化ルーチンも準備してある．M 凸と $M^\natural$ 凸についても同じである．

ODICON は C 言語によるオープンソースソフトウェアであり，単体での活用はもちろん，別のプログラムに組み込まれることも想定している．最小化したい離散関数をもつ利用者は，その関数を C 言語上の関数として記述して，このソルバのルーチンを呼び出せばよい．一連のルーチンは，アルゴリズムの素直な実装を目指し，入出力インタフェースも自然なものになるように留意した．

3.1 実装ルーチンの構成

ODICON には，表 1 のようなルーチンがある．以下はそのうちの 1 つである[2]．

```
double mgconv_minimize(
            int dim,
            double f(int dim, int x[]),
            int init[],
            int lower[],
            int upper[]);
```

これは，`dim` 次元の $M^\natural$ 凸関数 `f()` を最小化するルーチンである．利用者は，最小化しようとする離散関数が L / $L^\natural$ / M / $M^\natural$ 凸性のうちどの離散凸性を有するかに従って，該当するルーチンを呼び出すことになる．上記の例では，ルーチンは初期解 `init[]` から探索を行い，最急降下法で最小解にたどりついて，その最小値を返す．そして初期解 `init[]` を上書きして，たどり着いた最小解を書き込む．最小解が複数あっても，得られるのはそのうちの一つだけである．探索範囲は `lower[`i`]`$\leq$`init[`i`]`$\leq$`upper[`i`]` $(0 \leq i <$`dim`$)$ に限定される．

上記のルーチンの第 2 引数には，最小化する離散 $M^\natural$ 凸関数を与える．この離散関数は次の型（出力が `double`，入力が `int` と `int` の配列）で宣言しておく．

```
double f(int dim, int x[]);
```

この関数を「関数（ルーチン）の引数」として渡すのだが，通常の C 言語プログラムではあまり使われない手法のため，一般の利用者にとっては難しく思えるか

表 1 ODICON に実装した離散凸関数最小化ルーチン

L 凸関数最小化	（局所探索の方法）
lconv_minimize	最急降下法 [1]（全列挙）
lconv_minimize_IFF	最急降下法 [1]（IFF）
lconv_minimize_FW	最急降下法 [1]（FW）
lconv_minimize_scaling	スケーリング法 [1]（全列挙）
lconv_minimize_scaling_IFF	スケーリング法 [1]（IFF）
lconv_minimize_scaling_FW	スケーリング法 [1]（FW）
lconv_minimize_relax	連続緩和法 [6]（IFF）
$L^\natural$ 凸関数最小化	（局所探索の方法）
lgconv_minimize	最急降下法 [1]（全列挙）
lgconv_minimize_IFF	最急降下法 [1]（IFF）
lgconv_minimize_FW	最急降下法 [1]（FW）
lgconv_minimize_scaling	スケーリング法 [1]（全列挙）
lgconv_minimize_scaling_IFF	スケーリング法 [1]（IFF）
lgconv_minimize_scaling_FW	スケーリング法 [1]（FW）
lgconv_minimize_relax	連続緩和法 [6]（IFF）
M 凸関数最小化	（局所探索の方法）
mconv_minimize	最急降下法 (i) [1]
mconv_minimize2	最急降下法 (ii) [5]
mconv_minimize3	最急降下法 (iii) [8]
mconv_minimize_scaling	スケーリング法 [5]
mconv_minimize_relax	連続緩和法 [7]
$M^\natural$ 凸関数最小化	（局所探索の方法）
mgconv_minimize	最急降下法 (i) [1]
mgconv_minimize2	最急降下法 (ii) [5]
mgconv_minimize3	最急降下法 (iii) [8]
mgconv_minimize_scaling	スケーリング法 [5]
mgconv_minimize_relax	連続緩和法 [7]

[2] 関数名の接頭辞の `mg` の g は general の意味で，M 凸の一般化である $M^\natural$ 凸を表している．

もしれない．しかし記述は簡単で，引数に関数名をそのまま書くだけでよい．なお，最小化したい離散凸関数としては，いずれのルーチンでもこの型を受け取るよう統一している．

例えば，次のような 3 次元の離散 $M^\natural$ 凸関数

$$f(x) = x_0^4 + (x_1 - 3)^2 + 5(x_2 - 7)^2$$

を C 言語で実装すると，次のようになる．

```
double f(int dim, int x[]) {
    double r = 0;

    assert(dim == 3);  /* check dim */
    r += x[0] * x[0] * x[0] * x[0];
    r += (x[1] - 3) * (x[1] - 3);
    r += 5 * (x[2] - 7) * (x[2] - 7);
    return r;
}
```

この関数を，原点から探索して最小化するには，別の関数で次のように処理する．探索範囲は $-100 \leq x_i \leq 100$ とする．

```
int main() {
    int x[3]     = { 0, 0, 0 };
    int lower[3] = { -100, -100, -100 };
    int upper[3] = {  100,  100,  100 };
    double min = mconv_minimize(3, f,
                     x, lower, upper);
    ...
```

このように呼び出しを行うと，`min` には最小値 `0`，`x[]` にはそれを実現する最小解 `{0,3,7}` が代入される．

このようなルーチンを，離散 L / $L^\natural$ / M / $M^\natural$ 凸関数の 4 種類の関数クラスの最小化に対して用意し，それぞれの関数クラスについて，複数の最小化アルゴリズム（最急降下法，スケーリング法，連続緩和法）を実装し利用者がアルゴリズムを指定できるようにした．ルーチンの引数はほぼ共通しており，呼び出すルーチン名を変更するだけで異なるアルゴリズムを試すことができる．

離散関数の性質とアルゴリズムの組合せを間違えると何が起こるかについては，一般論としては残念ながら「動作が保証されない」としか言えない．わかっていることは，正しい組み合わせで正しい答えが出ることだけであって，間違ったアルゴリズムからは，運良く正しい答えが出るかもしれないし，間違った答えが出るかもしれないし，あるいは途中でアルゴリズムが止まって答えが出ないかもしれない．ケースバイケースである．ある関数が離散 L / $L^\natural$ / M / $M^\natural$ 凸性を満たすかどうかの判定ができればよいのだが，一般には探索前にも後にもこの判定を行うことは難しい．幸運な例外が 2 次関数で，簡単なルールで探索前に判定ができる（第 4 節参照）．

なお，離散凸関数最小化アルゴリズムの実装にあたっては，劣モジュラ集合関数最小化や，連続凸関数最小化のルーチンも必要になる．これらには以下のプログラムを利用したが，ルーチンの呼び出し方が統一感を持つように変換を施すインタフェースも整備した．

- 劣モジュラ関数最小化に，Iwata–Fleischer–Fujishige (IFF) アルゴリズム [3] の岩田氏による実装．
- 劣モジュラ関数最小化に，Fujishige–Wolfe (FW) アルゴリズム（最小ノルム基底法）の藤重氏・磯谷氏 [2] による実装．
- 連続関数最小化に，quasi-Newton 法の J. Nocedal 氏による実装である ‘L-BFGS’[3] と，工藤氏による C++言語へのラッパー[4]．
- 乱数を生成するために，斎藤氏・松本氏による ‘SIMD-oriented Fast Mersenne Twister’[5]．

表 1 に示したルーチンについて説明する．

- L / $L^\natural$ 凸関数に対する最急降下法には，局所探索の劣モジュラ集合関数最小化に (i) 全列挙，(ii) IFF，(iii) FW を用いた 3 種類のルーチンを用意している．IFF や FW は（関数値評価のほかに）実数計算を含むため丸め誤差の影響を受けるが，全列挙は（多項式時間アルゴリズムではないものの）より安定している．劣モジュラ集合関数最小化において，IFF は初期点と最小解が近いときに早く終了するが，FW の実行時間は初期点と最小解の距離にあまり依存しないことが観察されている．
- L / $L^\natural$ 凸関数に対するスケーリング法においても，局所探索の劣モジュラ集合関数最小化に (i) 全列挙，(ii) IFF，(iii) FW を用いた 3 種類のルーチンを用意している．
- L / $L^\natural$ 凸関数に対する連続緩和法において，仕上げ段階に用いる最急降下法には IFF に基づく最急降下法を採用した．これは，連続緩和解を整数

[3] http://www.ece.northwestern.edu/∼nocedal/lbfgs.html
[4] http://chasen.org/∼taku/software/misc/lbfgs/
[5] http://www.math.sci.hiroshima-u.ac.jp/∼m-mat/MT/SFMT/

に丸めた初期点が真の最小解に十分近いことが多く，そのような場合にはIFFは非常に早く終了するという経験的事実を考慮した結果である．

- M / M$^\natural$ 凸関数に対する最急降下法としては，第2.1節の最後に述べた3種類のアルゴリズム：
  (i) $f(x-e_i+e_j)$ を最小化する (i,j) を見出す基本形 [1],
  (ii) 任意に選んだ i に対して $f(x-e_i+e_j)$ を最小化する j を見出す修正形 [5],
  (iii) 修正形 (ii) に領域縮小を組み込んだもの [8],
  のそれぞれに対応するルーチンを用意している．
- M / M$^\natural$ 凸関数に対するスケーリング法は，Moriguchi–Murota–Shioura [5] のアルゴリズムである．これはスケーリング後もM / M$^\natural$ 凸性が保たれるようなM / M$^\natural$ 凸関数についてのみ計算量を保証するアルゴリズムとして提案されたものであるが，一般のM / M$^\natural$ 凸関数に対しても（計算量の理論保証はないものの）真の最小解を出力する．
- M / M$^\natural$ 凸関数に対する連続緩和法において，仕上げ段階に用いる最急降下法には，M / M$^\natural$ 凸関数の最急降下法の (ii) のルーチンを用いている．ほかの2つの最急降下法を使っても大きな差はない．

3.2 使用法

動作環境

ODICONはC言語ソースの形で配布しているので，利用するにはCコンパイラが必要である．開発にはANSI C (C89) 規格に準じたもの[6]を用いたので，昨今のCコンパイラならほぼ問題ないと思われる．Windowsではcygwin上のgcc，Mac OS XやLinuxならばOSに付属する開発環境のgccが安心である[7]．

コンパイル

ダウンロードした `odicon-20YYMMDD.tar.gz` を適当なディレクトリで展開する．`20YYMMDD` はバージョン番号を表す（公開した日付でもある）．

```
% gzip -cd odicon-20YYMMDD.tar.gz | tar xvf -
```

環境に合わせて `Makefile` を編集し，'make' コマンドを実行する．

```
% cd odicon-20YYMMDD
% make
```

この作業で，サンプルである `odicon` という実行ファイルが生成される．これはテスト用の関数を最小化するプログラムである．使い方は次のとおりである．

```
% ./odicon L dim [seed]
% ./odicon M dim [seed]
```

引数1 'L' か 'M' を指定する．
引数2 インスタンスの次元数を指定する．
引数3 （省略可）インスタンス生成に使う乱数の種を指定する．

このサンプルプログラムは指定された引数に従ってインスタンスを乱数で生成し，いくつかのアルゴリズムで最小化を行う．

自作プログラムの作り方

上記の使い方を踏襲して，自作プログラムの作り方を説明する．ODICONの配布時点ではmain( ) 関数が `odicon.c` に含まれているが，それを使わずに，自前のプログラムでmain( ) を用意したい．その例として，`sample.c` というプログラムを収録している．このプログラムを元に，`hogehoge` という名前のプログラムを作ることにする．それには，`Makefile` 中の

```
PACKAGE = odicon
```

という行を

```
PACKAGE = hogehoge
```

と書き換える．そして `sample.c` を `hogehoge.c` という名前でコピーする．

これだけで，もっともシンプルな使い方，つまり，次の2つの使い方ができる．

- 'make' を実行すると実行ファイル `hogehoge` が生成される．
- 'make tar' を実行すると，必要なソースが `hogehoge-20YYMMDD.tar.gz` に固められる．

この後，`hogehoge.c` に含まれるmain( ) 関数を自由に書き換えればよい．

4. Webアプリケーション

われわれの研究グループでは，前述の離散凸関数最小化ソルバODICONを始めとするアプリケーションソフトウェアを公開すると同時に，ソルバの動作を手軽に試せるように，いくつかはWeb上のデモンストレーションソフトウェアとしても公開している [4]．ここでは，そのなかから，離散凸関数最小化ソルバをい

[6] gcc 4.1.2 に `-std=c89` のオプションをつけた．
[7] Windows の Visual Studio のコマンドラインコンパイラ cl でも動作すると思われる．

くつか取り上げる．

2 次関数

$L^\natural$ 凸 2 次関数最小化のデモンストレーションにおいては，利用者が定義する 2 次の $L^\natural$ 凸関数

$$f(x) = \frac{1}{2}x^\top Ax + b^\top x$$

の最小化問題を最急降下法で解く．利用者は，n 次対称行列 $A = (a_{ij})$ と n 次元ベクトル $b = (b_i)$ を入力することによって関数 f を定義し，初期解も指定することができる．アプリケーションは，利用者の入力が $L^\natural$ 凸関数を与えるかを判定する[8]（図 1 の上）．入力が正しい場合（あるいは利用者が正しいものに修正した場合）には，「Minimize」ボタンを押すことが可能になる．利用者がこのボタンを押すと，アプリケーションは（全列挙に基づく）最急降下法を適用し，計算の途中経過と共に出力する（図 1 の下）．$M^\natural$ 凸 2 次関数に関しても，同様である．

擬分離 $L^\natural$ 凸関数

擬分離 $L^\natural$ 凸関数最小化のデモンストレーションにおいては，

$$f(x) = \sum_{i \neq j} f_{ij}(x_i - x_j) + \sum_{i=1}^{n} f_i(x_i)$$

の形の離散凸関数（擬分離 $L^\natural$ 凸関数）の最小化問題を最急降下法で解く．各 f_{ij}, f_i は 1 変数の凸関数であり，2 次関数，4 次関数，指数関数，絶対値関数などを提供している．利用者は，パラメータを入力することによって関数 f を定義し，初期解も指定することができる．アプリケーションは，(IFF 法に基づく）最急降下法を適用し，計算の途中経過と共に出力する．

層型 $M^\natural$ 凸関数

層型 $M^\natural$ 凸関数最小化のデモンストレーションにおいては，

$$f(x) = \sum_{Y \in \mathcal{T}} f_Y(x(Y)), \qquad x(Y) = \sum_{i \in Y} x_i$$

の形の離散凸関数の最小化問題を最急降下法で解く．ここで，$\mathcal{T}$ は層族（任意の $X, Y \in \mathcal{T}$ に対して $X \cap Y$, $X \setminus Y$, $Y \setminus X$ のどれかは空集合である集合族）で，各 f_Y は 1 変数の凸関数である．2 次関数，4 次関数，指数関数，絶対値関数などを提供している．利用者は，パラメータを入力することによって関数 f を定義し，初期解も指定することができる．アプリケーションは，最急降下法 (i) を適用し，計算の途中経過と共に出力する．

[8] f が $L^\natural$ 凸関数 $\iff$ $a_{ij} \leq 0$ $(i \neq j)$, $\sum_{j=1}^{n} a_{ij} \geq 0$ $(i = 1, \ldots, n)$.

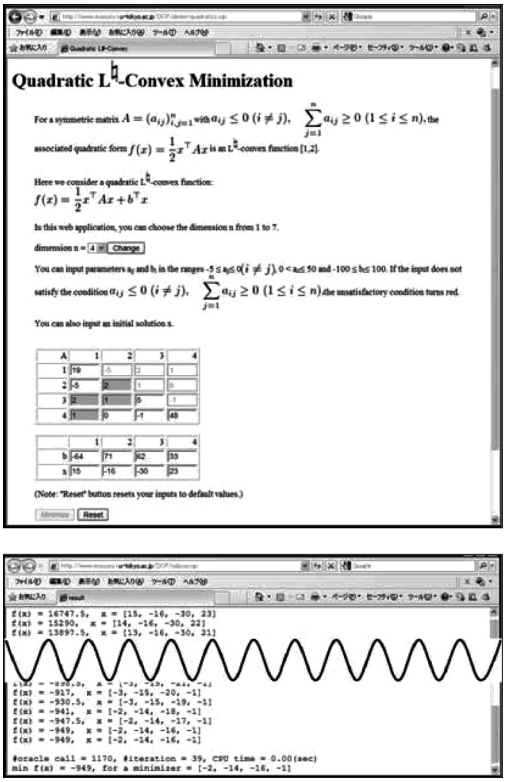

図 1 開発したオンラインソルバ（$L^\natural$ 凸 2 次関数最小化問題の入力と出力）

その他の Web アプリケーション

もっと実際的な問題を扱った Web アプリケーションも公開している．「在庫管理」と「コールセンターにおけるシフトスケジューリング」について，本特集の森口氏の記事を参照されたい．

5. 終わりに

ODICON の開発を通じて得た知見を書き並べてみたい．

- 今さらではあるが，劣モジュラ関数最小化のルーチンによって性能に大きな違いがあることを，身をもって体験した．指数時間アルゴリズムから，多項式時間アルゴリズム（IFF や FW）に切り替えたときの性能差は衝撃であった．もちろん頭では「指数時間アルゴリズムでは解けないに等しい」ことはわかっているが，体験してありがたみが身に染みた．この経験は貴重であった．
- M / $M^\natural$ 凸関数は L / $L^\natural$ 凸関数に比べて簡単で，現実的に解ける問題のサイズでいうと，1000 次元と 100 次元ぐらいの違いがある．M / $M^\natural$ 凸関

数と L / L$^\natural$ 凸関数は双対の関係にあるから，実用上は似た難しさにあることを期待していた．実際には（どちらかというと，理論どおりに）解きやすさが全く違うということを体験した．

- M / M$^\natural$ 凸関数最小化は比較的簡単と言えども，アルゴリズムによって計算量のオーダーが違う．そのことがプログラムの実行速度にも有意な差となって現れた．多項式時間アルゴリズムと言えども，理論上の改善は実用上の性能向上にきちっとつながっていることがわかった．

M / M$^\natural$ 凸関数最小化のプログラムを自力で作ろうとすると，簡単なアルゴリズムでも性能が出て，100 次元くらいが解けるようになるかもしれない．しかし L / L$^\natural$ 凸関数を同じように解くと 10 次元ほどがやっとで，実用的な問題が全く解けない．ODICON を使っていただければ，どちらも 10 倍ほど大きな規模の問題が解けるようになる．離散凸関数を手に入れられたときには，ぜひご活用いただきたい．また，利用された場合は，次の開発や研究のモチベーションにもなるので，うまく解けたのか解けなかったのかなどをお知らせいただけるとありがたい．

参考文献

[1] 室田一雄，離散凸解析，共立出版, 2001.

[2] S. Fujishige, and S. Isotani, A Submodular Function Minimization Algorithm Based on the Minimum-norm Base, *Pacific Journal of Optimization*, **7**, 3–17, 2011.

[3] S. Iwata, L. Fleischer, and S. Fujishige, A Combinatorial Strongly Polynomial Algorithm for Minimizing Submodular Functions, *Journal of ACM*, **48**, 761–777, 2001.

[4] K. Murota, S. Iwata, A. Shioura, S. Moriguchi, N. Tsuchimura, and N. Kakimura, DCP (Discrete Convex Paradigm), `http://www.misojiro.t.u-tokyo.ac.jp/DCP/`

[5] S. Moriguchi, K. Murota, and A. Shioura, Scaling algorithms for M-convex Function Minimization, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, **E85-A**, 922–929, 2002.

[6] S. Moriguchi, and N. Tsuchimura, Discrete L-convex Function Minimization Based on Continuous Relaxation, *Pacific Journal of Optimization*, **5**, 227–236, 2002.

[7] S. Moriguchi, A. Shioura, and N. Tsuchimura, M-convex Function Minimization by Continuous Relaxation Approach—Proximity Theorem and Algorithm, *SIAM Journal on Optimization*, **21**, 633–668, 2011.

[8] A. Shioura, Fast Scaling Algorithms for M-convex Function Minimization with Application to the Resource Allocation Problem, *Discrete Applied Mathematics*, **134**, 303–316, 2003.

[9] 土村展之，ODICON, `http://www.misojiro.t.u-tokyo.ac.jp/~tutimura/odicon/`

[10] 土村展之，森口聡子，室田一雄，離散凸最適化ソルバとデモンストレーションソフトウェア，応用数理学会論文誌，第 23 巻 2 号，2013，掲載予定．