2024 年 7 月現在

https://www.midaco-solver.jp

ユーザーマニュアル

Version 6.0

概要

MIDACO は、単目的および多目的最適化のための高性能な数理最適化ソルバーです。汎用ソフトウェアとして構築された MIDACO は、様々な最適化問題に利用できます。MIDACO は微分が不要（derivative-free）なハイブリッド進化的アルゴリズムに基づいており、目的関数と制約関数は、非線形性、非凸性、不連続性、確率的ノイズなど、難易度の高い関数特性も許容するブラックボックスとして扱われます。決定変数には、連続変数、離散変数（2 値変数など）、またはその両方（混合整数という）を用いることができます。また、MIDACO には（強力な）並列化オプションが用意されており、 1 回の評価に時間を要するような、CPU 時間が膨大に必要な事例（数値シミュレーションなど）には特に効果的です。

クイックメニュー

目次

MIDACO の強み

- **大域的最適化（Global Optimization） ソルバーとして**
  - 単目的最適化と多目的最適化に対応
  - 連続変数、離散／組合せ変数、混合整数変数に対応
  - 制約条件あり・なし両方の問題に対応

- **ハイブリッド進化的アルゴリズムを搭載**
  - 基本となるアルゴリズムはメタヒューリスティックなアントコロニー最適化（ACO）
  - バックトラック法を用いた内部ハイブリッド化による局所収束の高速化
  - 目的関数と制約関数は、線形でも非線形でもよい (微分可能性は必須ではない)
  - ブラックボックスソルバーなので、目的関数や制約関数が不明でもよい（たとえば、シミュレーションなどの場合）

- **大規模な問題にも対応**
  - 最大で**100,000**個の変数に対応
  - 最大で数千個の制約条件と数百個の目的関数に対応

- **並列化に対応**
  - 複数の言語においてさまざまな並列化スキームを提供
  - 数千コア／スレッドの大規模並列処理 が可能 (GPGPU を含む)

- **多くのプログラミング言語に対応**
  - Excel、VBA、Java、C#、R、Matlab、Octave、Python、Julia、C/C++、Fortran など

- **ソースコードについて**
  - **10 年以上** におよぶ徹底的なテストと継続的な改良
  - 超軽量 (200kb 未満) の圧縮 ANSI-C コード (*midaco.c*) または Fortran コード (*midaco.f*)
  - 完全自給自足のソースコード (サードパーティ製ソフトウェア等への依存なし)
  - Win ／ Mac ／ Unix、Web サーバなど、すべてのプラットフォームでコンパイルして実行可能
  - **非常に扱いやすく、組み込みも簡単**

- **最高成績を記録**
  - MIDACO は惑星間軌道のベンチマークで複数の最高成績を記録

- **開発環境について**
  - 欧州宇宙機関 (ESA) との共同開発
  - 宇宙航空研究開発機構 (JAXA) の支援により改良

はじめに

MIDACO は、数理最適化を実行するためのソフトウェアツールです。MIDACO のアルゴリズムは、**単目的**および **多目的**最適化問題のための**汎用ソルバー** として構築されています。MIDACO の特徴は、(制約条件のある) 混合整数非線形計画 (MINLP) 問題を扱える点にあります。混合整数 *mixed integer* とは、決定変数に連続型 (1.23 や 4.56 など) と離散型 (1、2、3 など) の両方が含まれる最適化問題を指します。MIDACO が想定する一般的な多目的 MINLP 問題の数学的定式化は、以下の通りです。

$$
\begin{aligned}
&\text{最小化} \qquad f_1(x),\ f_2(x),...,\ f_O(x) \\
&\\
&\text{条件} \qquad g_i(x) \ = \ 0, \quad i \ = \ 1,...,m_e \\
&\qquad\qquad\ \ g_i(x) \ \geq \ 0, \quad i \ = \ m_e+1,...,m \\
&\qquad\qquad\ \ x_l \ \leq \ x \ \leq \ x_u \ \ \text{(ボックス制約)}
\end{aligned}
$$

上記の定式化では、ベクトル $f_{1,...,O}(x)$ が目的関数、ベクトル $g_{1,...,m}(x)$ が制約条件 を表しています。すべての目的関数は、一般性を失うことなく、最小化の対象となります。制約条件ベクトル $g(x)$ の最初の値 $1,...,m_e$ は等式制約を表しており、残りの値 $m_e+1,...,m$ は不等式制約を表します。決定変数のベクトル x に、連続変数と離散変数 (整数変数、カテゴリー変数、組み合わせ変数ともいう) の両方が含まれる場合、最初に連続変数が、その後に離散変数が格納されます。さらに、決定変数 x は、下限値 lower bounds である x_l と上限値 upper bounds である x_u のボックス制約 box constraints を取ります。

MIDACO は、上記のような多目的 MINLP 問題を解くために、拡張進化的アントコロニー最適化 *Ant Colony Optimization* (ACO) [34] アルゴリズムと、制約条件処理を行うオラクルペナルティ法 *Oracle Penalty Method* [36] とを組み合わせています。MIDACO の ACO アルゴリズムは、いわゆるマルチカーネル・ガウス確率密度関数 (PDF) に基づいており、反復子 *iterates* (蟻 *ants* または個体 *individuals* ともいう) の標本を生成します。整数の決定変数に対しては、PDF の離散化された形式が適用されます ([34] 参照)。図 1 は、連続領域 (左) と整数領域 (右) における 3 つのカーネル PDF を伴うガウス PDF を示しています。

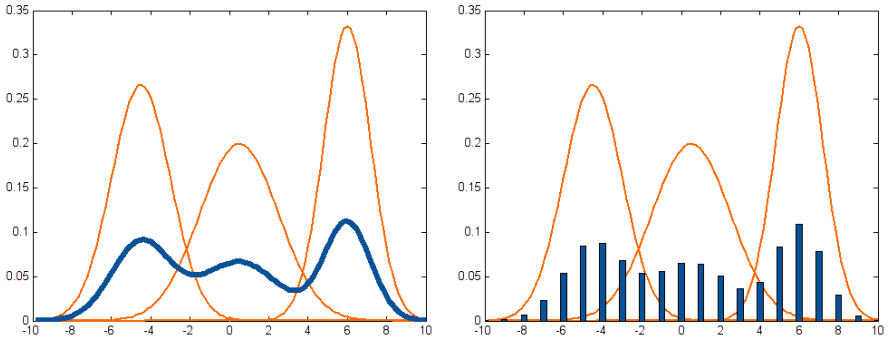


図1　連続領域 (左) と整数領域（右）のマルチカーネル・ガウス PDF

MIDACO の内部で制約条件を処理するのは、（ACO、GA、PSO などの）メタヒューリスティック探索アルゴリズムのために開発された先端的手法であるオラクルペナルティ法 Oracle Penalty Methodです。この手法は、目的関数 $f(x)$ に対応するオラクル *Oracle*（[36] では「オメガ *Omega*」）と呼ばれるパラメータを用いて**大域的最適解 global optimal** を求めることを目指します。この手法は自己適応的 self-adaptive であり、MIDACO も自己適応アルゴリズム self-adaptive algorithm (SAA) に分類されます。図2は、目的関数値 $f(x)$ と、$g(x)$ の制約条件違反を表す残差値 *residual* value である $res(x)$ に依存する拡張オラクルペナルティ関数の形状を示しています (一般的に使用されるL_1 ノルムで測定)。

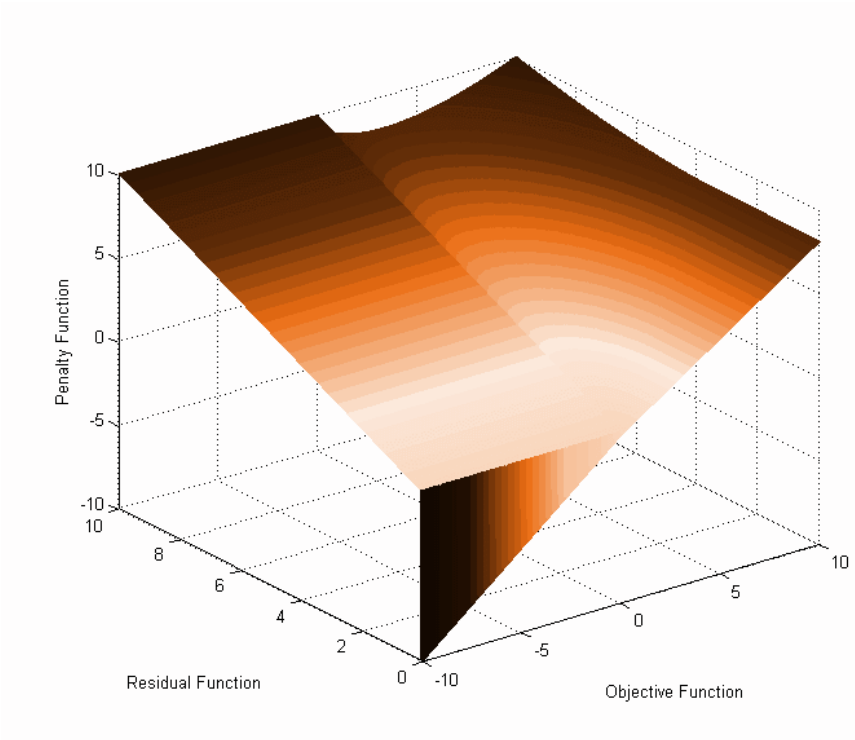


図2　拡張オラクルペナルティ関数の形状

他の大多数の進化的最適化アルゴリズムと同様に、MIDACO は、目的関数 $f(x)$ と制約条件 $g(x)$ をいずれも **ブラックボックス black-box** 関数とみなします。つまり、ある入力ベクトル x に対して MIDACO が確認するのは、目的関数 f(x) と制約関数 g(x) の戻り値のみです。MIDACO では、目的関数と制約関数の実際の計算方法についての特別な知識を必要としません。そのため、目的関数と制約関数については、（高い）非線形性や、非凸性、非平滑性、非微分性、不連続性などの関数特性のほか、確率ノイズ stochastic noise といった難しい特性までもが許容されるのです。

こうした **ブラックボックス** のコンセプトにより、ユーザーは必要に応じて**まったく自由に** 問題を定式化することができます。たとえば、問題を設定する際に、*if* 文やサブルーチンの呼び出しなど、あらゆる種類のプログラミング文を含めることが可能であり、外部のプログラム (Simulink やText-I/Oなど) を呼び出すことも可能です。

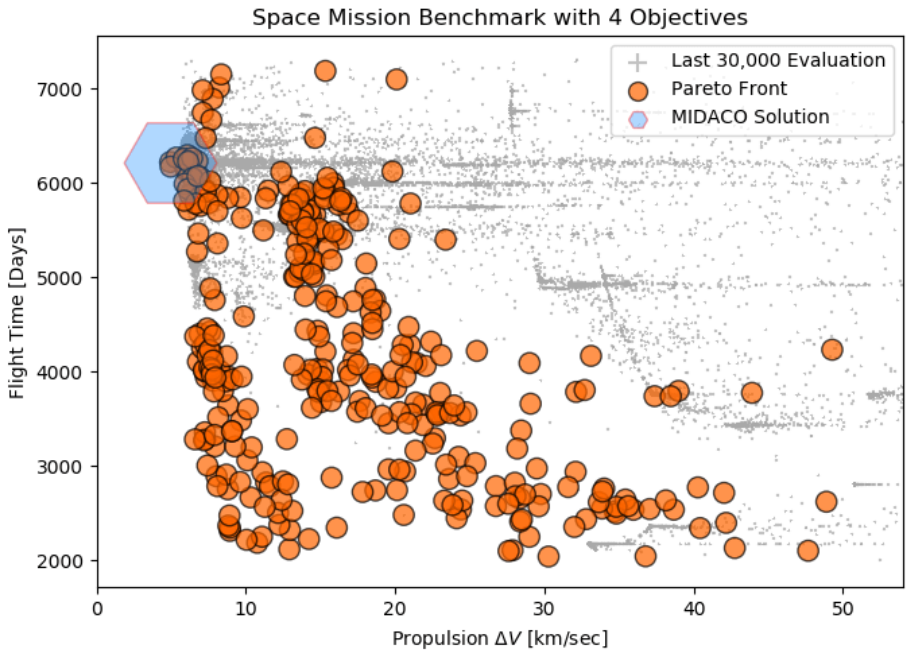
全体的なパフォーマンスを向上させるために、MIDACO は多くのヒューリスティックを実装 しており、内部では迅速な局所収束を目指して、擬似勾配に基づくバックトラック直線探索と連動しています。しかし、他のヒューリスティックなアルゴリズムと同様に、MIDACO も大域的最適解への到達を保証するものではありません。MIDACO が目指すのは、複雑な実世界での利用において、合理的な時間で合理的に優れた解決に向けた最適化を可能とする、堅牢なソルバーであることです。
大規模な MINLP ベンチマーク問題のセットに対して、MIDACO が高速かつ信頼性の高い大域的な最適解を得られることは、多様な数値テストで示されています ([37]、[36] 、および特に [44] を参照)。MIDACO と既存の決定論的 MINLP アルゴリズムとの数値的な比較は [37] を、MIDACO と確率的探索アルゴリズム（遺伝的アルゴリズム、散布図、変数近傍、共分散行列ベースの探索を含む）との数値比較は [34] や [35] 、[40] を参照してください。
MIDACO で解くことのできる一般的な大域的最適化問題（Rosenbrock、Ackley、Rastringin など有名な NLP ベンチマークを含む）の一覧は、MIDACO ウェブサイトのベンチマークページで利用できます。MIDACO ウェブサイトのベンチマークページで検討されている MIDACO の実行時間と性能（目的関数・変数・制約条件の数など）は、**進化的コンピューティングの最先端** であり、特に MINLP と数千の変数を許容する大規模最適化のパフォーマンスにぜひご注目ください。

汎用ソルバーである MIDACO は、さまざまな最適化問題に利用できます。以下、MIDACO ウェブサイトの利用事例ページから一部を紹介すると、惑星間空間の軌道設計 [18]・[38]・[41]・[42]、ロケットの建設と制御 [39]、衛星配置操作 [45]、低推力軌道変換最適化 [5]、キューブサット展開軌道設計 [26]、航空機の枠構造最適化 [50]、航空機バッテリー最適化 [4]、乗客と航空機の割り当て [30]、クワッドローターの姿勢制御 [19]、化学工場の配置 [35]・[40]・[16]、廃水処理 [35]、水供給ネットワーク [11]、カメラの最適配置 [20]、土壌パラメータ最適化 [48]、メタマテリアル・ファブリック・デザイン [14]、金融における distance-to-default モデル [3]、売上予測 [13]、ワイヤレスネットワーク通信 [2]・[6]・[7]・[8]・[9]・[10]・[29]、ネットワークの構造的脆弱性 [17]、潜水艦の構造最適化 [51]、バイオ技術におけるパラメータ最適化 [32]、神経科学 [25]、高調波低減フィルタ設計 [22]・[23]、林業予測 [12]、経済分析 [47]、自動車ベルトコンベアの最適化 [28]、自動車産業計画 [1]・[15]、牽引用電気駆動装置 [53]、複合発電所の最適化 [27]、CO_2 サイクル発電設計 [52]、低炭素エネルギー構成の最適化 [49]、油田コントロール [31]、天然ガスプラントの最適化 [46] などが挙げられます。

CPU 時間のかかる 問題 (目的関数および／または制約関数を 1 回評価するだけでかなりの時間を要する問題) に対しては、MIDACO が効率的な並列化ストラテジーを提供します。MIDACO では、複数の解答候補を**並行して**評価できます。この方法は「同時評価 *co-evaluation* 」または「細粒度 *fine-grained* 」並列化ともいい、最適化全体の所要時間を大幅に短縮することができます。MIDACO の並列化ストラテジーは、数千のスレッド/コアまで拡張可能な、堅牢かつポータブルなコンセプトであるリバースコミュニケーション reverse communication により実装されています。このコンセプトのおかげで MIDACO は、Fortran、C/C++ (openMP、openMPI、GPGPU)、Matlab (parfor)、R(dopar)、Java (Fork/Join)、Python(multipro-cessing、mpi4py、spark) など、様々な並列化スキームに対応した複数のプログラミング言語を通じて、その並列化ストラテジーを提供することができます。MIDACO の並列化ストラテジーによる簡単な実行例のテンプレートは、MIDACO ウェブサイトの並列化ページでご覧ください。

多目的最適化問題 に対して、MIDACO はユートピア・ネイディア・バランス *Utopia-Nadir-Balance* [43] という新たなコンセプトを用いています。このコンセプトは、特に（航空）宇宙分野で発生する多数目的 *many-objective* 問題のために開発されました（多数目的 many-objective 最適化問題は、4 つ以上の目的関数を含むという点で、多目的 multi-objective 最適化問題とは区別されます)。ユートピア・ネイディア・バランスのコンセプトでは、多目的問題への従来のアプローチとは異なり、パレートフロントの特定領域にアルゴリズムの探索を集中させます。デフォルトでは、パレートフロントの中央 *central*（または中間 *middle*）部分に探索が集中します。この部分が、すべての目的関数のなかで**最もバランスのとれた**トレードオフを提供するからです。ただし、**BALANCE** パラメータを調整すれば、ユーザーは自由にパレートフロントの他の部分に焦点を変更することもできます。

ユートピア・ネイディア・バランス *Utopia-Nadir-Balance* の主な利点は、パレートフロントのすべての部分を等しく重要視する従来の方法（非優越ソート *non-dominated sorting* など）に比べて、パレートフロントの最も望ましい部分を、通常よりもより早く発見し、より深く探索できる ことにあります。ユートピア・ネイディア・バランスのコンセプトは、凸型、凹型、混合型、分離型など、あらゆる種類のパレートフロントに対応しており、最大で数百個の目的関数を設定したテストに成功しています (MIDACO ウェブサイトのベンチマークページを参照)。下の図は、ESA の宇宙ミッションである Cassini1[18] のベンチマークに MIDACO を利用したプロットです。下のプロットでは、最初の目的関数 (推進力 ΔV を x 軸に表示) にバランスが設定されており、最後の 3 万回の評価で、MIDACO がパレートフロントの該当部分に探索を集中させる様子を観察できます。



このユーザーマニュアルの目的は、MIDACO を利用した最適化問題の設定と解決方法について、実践的なガイドラインを提供することです。MIDACO に搭載されている ACO アルゴリズムの理論的な詳細に興味のある方は、他の文献 (たとえば [34] や [35] など) を参照してください。また、オラクルペナルティ法の開発と特性に関する詳細な情報は、文献 [36] やオラクルペナルティ法のウェブサイトに掲載されています。

1 最適化問題

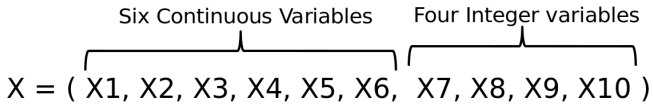
このセクションでは、最適化問題が MIDACO にどのように提示されるかについて説明します。MIDACO のアルゴリズムは、最適化問題を最も基本的な形式として、すなわち、ある入力変数 X に対して、対応する目的関数値 F(X) と制約関数値 G(X) を返す、ブラックボックスとして考えます。このブラックボックスのコンセプトは、進化的アルゴリズムでは一般的に用いられており、サブルーチンコールや外部のサブプログラムを含めて、目的関数や制約関数の値を定義する形式や計算方法を、ユーザーが自由に決定できます。ブラックボックスのコンセプトにより、目的関数や制約関数が難しい特性（高い非線形性、不連続性、確率的ノイズなど）を持つことや、実際の数学的定式が不明であること（シミュレーションコードなど）も許容されます。

連続変数と離散 (整数) 変数が同時に存在する混合整数問題の場合、連続変数は 変数のベクトル X に最初に格納され、離散変数はX の最後に格納 されます。また同様に、制約ベクトル G(X) での等式制約と不等式制約の場合も、ベクトル G に等式制約を最初に、不等式制約を最後に格納することで区別します。
例として、下のような制約条件付き混合整数問題を考えてみましょう。

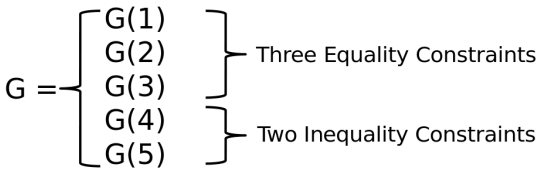
N	=	10	M	=	5
NI	=	4	ME	=	3

N : 変数の数 (合計)
NI : 整数変数の数
M : 制約条件の数 (合計)
ME : 等式制約の数

このとき、X については連続変数 Continuous Variables と整数変数 Integer Variables が次のように区別されます。



また G について、等式制約 Equality Constraints と不等式制約 Inequality Constraints は次のように区別されます。



どのような問題でも、決定変数 X の下限値と上限値 (XL と XU) を決める必要があります。開始点 X（初期解または初期点ともいう）も決めなければなりませんが、これは XL と XU の範囲内にある任意の点 (決定変数 X のベクトル) で構いません。MIDACO のデフォルトでは、すべてのサンプルで下限値が開始点になっています。MIDACO は探索空間全体を探索するので、一般的には (XL と XU で定義される) 探索空間をできるだけ小さくしておくことをお勧めします（ヒント：探索空間を縮小できる箇所を確認するためには、バウンズ・プロファイル BOUNDS-PROFIL をご利用ください)。それに対して開始点は、MIDACO では多くの場合で重要ではありません。

開始点 X が下限値または上限値を外れている場合、MIDACO はそれぞれ IFLAG=204 または IFLAG=205 というエラーメッセージを表示します。整数変数の場合は、対応する下限値と上限値も離散的な値である必要があります。開始点 X に整数として宣言された変数があり、それらの変数が連続値 (たとえば 0.123) である場合、MIDACO はエラーメッセージ IFLAG=881 を表示します。整数変数の境界値が連続値 (たとえば 0.123) の場合、MIDACO はエラーメッセージ IFLAG=882 または IFLAG=883 を表示します。

重要：**MIDACO は変数の整数型／離散型を常に尊重 します**。 MIDACO は、整数型で宣言された X 変数に対しては、問題関数の評価に連続値を用いません。これは、MIDACO アルゴリズムを旧来の MINLP アルゴリズム（*Branch & Bound* など) から区別する重要な機能です（旧来のアルゴリズムでは、最適化の過程で一時的に整数型から連続型に変更する、いわゆる整数変数の緩和 *relaxation* が必要です)。

1.1 問題の次元、限界値、開始点

このサブセクションでは、最適化問題の次元、下限値、上限値、決定変数ベクトル X の開始点を宣言する方法を説明します。問題の次元は、F・G・X の配列のサイズを参照します。下の *example_MINLPc.m* の Matlab スクリーンショットの問題設定では、目的関数 objectives が 1 つ、変数 variables が 4 つ（そのうち 2 つは整数 integer（離散）型）で示されています。さらに、制約条件 constraints が 3 つあり、そのうちの 1 つは等式制約 equality constraints です。

```
% STEP 1.A: Problem dimensions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
problem.o  = 1; % Number of objectives
problem.n  = 4; % Number of variables (in total)
problem.ni = 2; % Number of integer variables (0 <= nint <= n)
problem.m  = 3; % Number of constraints (in total)
problem.me = 1; % Number of equality constraints (0 <= me <= m)

% STEP 1.B: Lower and upper bounds 'xl' & 'xu'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
problem.xl = [ 1, 1, 1, 1];
problem.xu = [ 4, 4, 4, 4];

% STEP 1.C: Starting point 'x'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
problem.x  = problem.xl; % Here for example: 'x' = lower bounds 'xl'
```

1.2 問題関数の呼び出し

このサブセクションでは、最適化問題に対応する関数の呼び出しについて説明します。上で説明したように、MIDACO と最適化問題の間で伝える必要があるのは、決定変数のベクトル X、目的関数のベクトル F(X)、制約値のベクトル G(X) の 3 つの要素だけです。MIDACO のウェブサイトで配布されているサンプル問題では、それらの問題の関数呼び出しは、プログラミング言語に応じて次のように示されます。

Matlab	:	[f, g] = problem_function(x)
Python	:	problem_function(x) (return f, g)
Julia	:	problem_function(x) (return f, g)
R	:	problem_function <- function(f,g,x)
C/C++	:	problem_function(*double* *f, *double* *g, *double* *x)
Fortran	:	PROBLEM_FUNCTION(F,G,X)
VBA	:	PROBLEM_FUNCTION_VB(X,F,G)
C#	:	blackbox(double[ ] f, double[ ] g, double[ ] x)
Java	:	blackbox(double[ ] f, double[ ] g, double[ ] x)

下の Matlab のスクリーンショット *example_MINLPc.m* は、problem_function の設定を示しています。ここでは、決定変数の入力ベクトル x に対して、単一の目的関数値 objective value である f(1) と制約関数値 constraint function values である g(1)・g(2)・g(3) の 3 つを生成するための計算が行われます.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%   OPTIMIZATION PROBLEM   %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ f, g ] = problem_function( x )

% Objective functions F(X)
  f(1) = (x(1)-1)^2 ...
       + (x(2)-2)^2 ...
       + (x(3)-3)^2 ...
       + (x(4)-4)^2 ...
       + 1.23456789;

% Equality constraints G(X) = 0 MUST COME FIRST in g(1:me)
  g(1) = x(1) - 1;
% Inequality constraints G(X) >= 0 MUST COME SECOND in g(me+1:m)
  g(2) = x(2) - 1.333333333;
  g(3) = x(3) - 2.666666666;

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%   END OF FILE   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

なお、F・G・X 各配列のベクトルインデックス i は、$i = 1$ で始まる言語（Matlab、R、Fortran）もあれば、$i = 0$ で始まる言語（C++,Python,Java など）もあるので注意してください。

1.3 追加の入出力引数の受け渡し

サンプルで示されている問題関数の呼び出しは、F・G・X ベクトルの伝達のみを行う最小限のものです。ユーザーが問題関数の名前の変更や、入出力引数の追加をしたい場合は、問題関数の呼び出しを自由に変更し、希望のレイアウトにすることができます。C/C++ や Fortran の場合、こうした変更はサンプルファイルのソースコードで直接行うことができます。他の言語 (Matlab や Python など) の場合は、言語固有のゲートウェイファイルのソースコードを変更する必要があります。問題関数の名前や引数は、ユーザーがプログラムを少し編集すれば簡単に変更できます。

以下は、問題関数の呼び出しを編集した Matlab コードの一例です。

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%   OPTIMIZATION PROBLEM   %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ f, g, EXTRA_OUTPUT ] = USER_MODEL_NAME( x, EXTRA_INPUT )

###### Do something with EXTRA_INPUT ######

% Objective functions F(X)
  f(1) = (x(1)-1)^2 ...
       + (x(2)-2)^2 ...
       + (x(3)-3)^2 ...
       + (x(4)-4)^2 ...
       + 1.23456789;

% Equality constraints G(X) = 0 MUST COME FIRST in g(1:me)
  g(1) = x(1) - 1;
% Inequality constraints G(X) >= 0 MUST COME SECOND in g(me+1:m)
  g(2) = x(2) - 1.333333333;
  g(3) = x(3) - 2.666666666;

###### Prepare EXTRA_OUTPUT ######

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%   END OF FILE   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

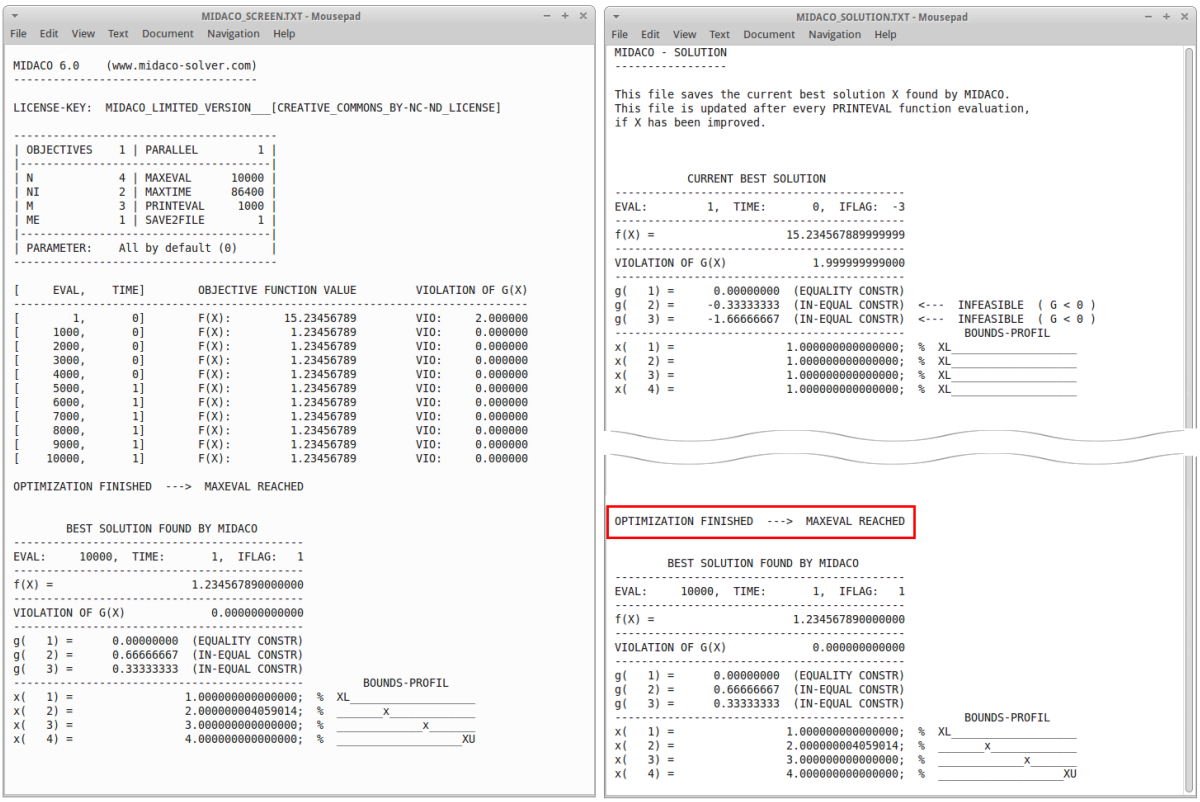
1.4 問題の実行を検証する

MIDACO（および他の最適化ソルバー）を使って特定の問題を計算する場合、その問題が正しく実装されていることが重要です。問題が適切に実装されているかを確認するために、最適化を実際に開始する前に、関数評価を 1 回のみ実行することが推奨されます。そこで、MIDACO の停止基準 Stopping Criteria オプションで MAXEVAL = 1 を設定します (セクション 3 を参照)。MAXEVAL=1 で設定すると、開始点を評価した後、MIDACO は直ちに停止し、対応する目的関数値と制約関数値を報告します。これにより、ユーザーは報告されたすべての目的関数値と制約関数値が妥当かどうかを手動でチェックできます。

2 MIDACO の計算画面と解

プリントを有効にすると、MIDACO は次の 2 つのテキストファイルを出力します。

MIDACO_SCREEN.TXT MIDACO_SOLUTION.TXT



```
MIDACO_SCREEN.TXT - Mousepad
File  Edit  View  Text  Document  Navigation  Help

MIDACO 6.0    (www.midaco-solver.com)
--------------------------------------

LICENSE-KEY:  MIDACO_LIMITED_VERSION___[CREATIVE_COMMONS_BY-NC-ND_LICENSE]

----------------------------------------
| OBJECTIVES    1 | PARALLEL         1 |
|--------------------------------------|
| N             4 | MAXEVAL      10000 |
| NI            2 | MAXTIME      86400 |
| M             3 | PRINTEVAL     1000 |
| ME            1 | SAVE2FILE        1 |
|--------------------------------------|
| PARAMETER:    All by default (0)     |
----------------------------------------

[     EVAL,    TIME]        OBJECTIVE FUNCTION VALUE         VIOLATION OF G(X)
------------------------------------------------------------------------------
[        1,       0]        F(X):         15.23456789        VIO:     2.000000
[     1000,       0]        F(X):          1.23456789        VIO:     0.000000
[     2000,       0]        F(X):          1.23456789        VIO:     0.000000
[     3000,       0]        F(X):          1.23456789        VIO:     0.000000
[     4000,       0]        F(X):          1.23456789        VIO:     0.000000
[     5000,       1]        F(X):          1.23456789        VIO:     0.000000
[     6000,       1]        F(X):          1.23456789        VIO:     0.000000
[     7000,       1]        F(X):          1.23456789        VIO:     0.000000
[     8000,       1]        F(X):          1.23456789        VIO:     0.000000
[     9000,       1]        F(X):          1.23456789        VIO:     0.000000
[    10000,       1]        F(X):          1.23456789        VIO:     0.000000

OPTIMIZATION FINISHED  --->  MAXEVAL REACHED


        BEST SOLUTION FOUND BY MIDACO
-----------------------------------------
EVAL:     10000,  TIME:       1,  IFLAG:   1
-----------------------------------------
f(X) =                    1.234567890000000
-----------------------------------------
VIOLATION OF G(X)             0.000000000000
-----------------------------------------
g(   1) =      0.00000000  (EQUALITY CONSTR)
g(   2) =      0.66666667  (IN-EQUAL CONSTR)
g(   3) =      0.33333333  (IN-EQUAL CONSTR)
-----------------------------------------                BOUNDS-PROFIL
x(   1) =                 1.000000000000000;  %  XL_________________
x(   2) =                 2.000000004059014;  %  _______x_________
x(   3) =                 3.000000000000000;  %  _____________x___
x(   4) =                 4.000000000000000;  %  _________________XU
```

```
MIDACO_SOLUTION.TXT - Mousepad
File  Edit  View  Text  Document  Navigation  Help
MIDACO - SOLUTION
-----------------

This file saves the current best solution X found by MIDACO.
This file is updated after every PRINTEVAL function evaluation,
if X has been improved.



           CURRENT BEST SOLUTION
-----------------------------------------
EVAL:         1,  TIME:       0,  IFLAG:  -3
-----------------------------------------
f(X) =                    15.234567889999999
-----------------------------------------
VIOLATION OF G(X)             1.999999999000
-----------------------------------------
g(   1) =      0.00000000  (EQUALITY CONSTR)
g(   2) =     -0.33333333  (IN-EQUAL CONSTR)  <---  INFEASIBLE  ( G < 0 )
g(   3) =     -1.66666667  (IN-EQUAL CONSTR)  <---  INFEASIBLE  ( G < 0 )
-----------------------------------------                BOUNDS-PROFIL
x(   1) =                 1.000000000000000;  %  XL_________________
x(   2) =                 1.000000000000000;  %  XL_________________
x(   3) =                 1.000000000000000;  %  XL_________________
x(   4) =                 1.000000000000000;  %  XL_________________

OPTIMIZATION FINISHED  --->  MAXEVAL REACHED

        BEST SOLUTION FOUND BY MIDACO
-----------------------------------------
EVAL:     10000,  TIME:       1,  IFLAG:   1
-----------------------------------------
f(X) =                    1.234567890000000
-----------------------------------------
VIOLATION OF G(X)             0.000000000000
-----------------------------------------
g(   1) =      0.00000000  (EQUALITY CONSTR)
g(   2) =      0.66666667  (IN-EQUAL CONSTR)
g(   3) =      0.33333333  (IN-EQUAL CONSTR)
-----------------------------------------                BOUNDS-PROFIL
x(   1) =                 1.000000000000000;  %  XL_________________
x(   2) =                 2.000000004059014;  %  _______x_________
x(   3) =                 3.000000000000000;  %  _____________x___
x(   4) =                 4.000000000000000;  %  _________________XU
```

MIDACO の計算画面ファイル（MIDACO_SCREEN.TXT）は、コンソールやコマンドウィンドウに表示される出力と同じです (Excel を除く)。MIDACO の計算画面と解ファイル（MIDACO_SOLUTION.TXT）のレイアウトは、基本的にすべてのプログラミング言語で共通ですが、言語によっては、F・G・X のベクトルインデックスとバウンズプロファイルに使用されるコメント記号に若干の違いがあります。MIDACO の計算画面と解ファイルで使用されるすべての略語について、次ページの表 1にまとめておきます。

表1: MIDACO の計算画面と解の出力ファイルで用いられる略語 (前ページの図2 つを参照)

OBJECTIVES	目的関数 objective functions の数
PARALLEL	並列処理される問題関数の呼び出し数 (同時評価 *co-evaluation* ともいう)
N	変数 variables の総数
NI	整数変数 integer variables の数　$0 \leq \mathrm{NI} \leq \mathrm{N}$
M	制約条件 constraints の総数
ME	等式制約 equality constraints の数　$0 \leq \mathrm{ME} \leq \mathrm{M}$
MAXEVAL	関数評価の最大回数 (停止基準、セクション 3を参照)
MAXTIME	実行に要する最大 CPU 時間割り当て (停止基準、セクション 3を参照)
PRINTEVAL	現時点の最良解のプリント頻度
SAVE2FILE	テキストファイル出力 [0= なし、1= あり、2= あり + 履歴ファイル作成]
PARAM	MIDACO 調整パラメータ (デフォルト = 0、セクション4を参照)
EVAL	実行された関数評価の回数
TIME	実行された CPU 時間・秒
F(X)	EVAL 評価後における目的関数の最良値
VIOLATION	制約条件の違反: ベクトル G に対するL_1 ノルムにより測定
IFLAG	MIDACO による情報フラグ（最終結果・警告・エラー）
F(i)	目的関数 F_i の各数値
G(i)	制約条件 G_i の各数値
X(i)	変数解 X_i の各数値

バウンズ・プロファイル BOUNDS-PROFIL は、X(i) の下限値 (XL(i)) と上限値 (XU(i)) に対する相対的な位置関係を図式化したものです。X(i) が下限値または上限値に 0.1% 以上近い場合、バウンズ・プロファイル BOUNDS-PROFIL にはそれぞれ大文字の「XL」または「XU」が表示され、そうでない場合は小文字の「x」が表示されます。

解ファイル（MIDACO_SOLUTION.TXT）には、計算画面にプリントされるすべての計算反復における X の数値が含まれます。つまり、PRINTEVAL で設定した関数評価の頻度で、解答ファイルが常に更新 されます。これは重要な機能で、ユーザーはプログラムの実行中であっても完全な解にアクセス可能で、最適化の実行が何らかの理由 (サーバーの再起動や停電など) で中断された場合の安全性を確保できます。さらに、最初の解（開始点、EVAL=1 ともいう）および最終結果が表示されます。解はその都度保存されます。

バウンズ・プロファイル BOUNDS-PROFIL は、解ファイルに含まれるすべての解について表示されます。目的関数 F(i) と制約条件 G(i) は、すべて個別に表示されます。制約条件が実行不可能である場合、「INFEASIBLE (G<0) 」(不等式制約の場合) もしくは「INFEASIBLE (G NOT=0) 」(等式制約の場合) と表示されます。
なお、プリントを繰り返しても、X が改善されない限り、解ファイルの X は更新されません。これは、解答ファイルが不必要に大きくなるのを防ぐためです。

2.1 PRINTEVAL と SAVE2FILE

PRINTEVAL は、計算画面における現時点の最良解の表示頻度を制御する、重要なパラメータです。このパラメータは、MIDACO アルゴリズムが実行される反復回数にはまったく関係しません。したがってユーザーは、表示に適した出力頻度になるよう、PRINTEVAL を自由に設定できます。PRINTEVAL に小さな値 (たとえば 10・123・500) を設定すると、出力頻度が高くなります。PRINTEVAL の値が大きい (たとえば 10000・100000・1000000) と、出力頻度は低くなります。
PRINTEVAL = 1 に設定すると、MIDACO が見つけた現時点の最良解が、評価のたびに表示されます。このオプションは、非常に多くの CPU 時間を必要とする問題や、デバッグ目的の場合にのみ有効です。一般的には、PRINTEVAL に 大きな値 を設定することをお勧めします。なぜなら、ユーザーは最適化の進捗状況をよりよく把握でき、(プリントコマンドの実行回数が少なくて済むため) MIDACO の動作も少し速くなるからです。実際に利用する際は、数秒から数分に一回だけ新しい行がプリントされるように PRINTEVAL を設定すれば十分です。

出力ファイルの作成は任意です。SAVE2FILE を 0 に設定すると、出力ファイルは作成されません。何も出力したくない場合 (たとえば、MIDACO を他のソフトウェア内部に埋め込み、最終解のベクトル X のみをそのソフトウェアでさらに数値処理する必要がある場合) は、PRINTEVAL を 0 に設定すると、視覚的な出力をすべて停止することができます。

重要：**PRINTEVAL=0 と SAVE2FILE=0 を設定すると、MIDACO は出力を完全に停止します**。

2.2 解履歴ファイル

MIDACO は、計算画面と解ファイルに加えて、評価されたすべての反復子 X とそれに対応する目的関数 F(X) および制約条件 G(X) の値の完全な解履歴 Solution History を作成できます。SAVE2FILE オプションに 1 よりも大きい値を指定すると、MIDACO は自動的に「MIDACO_HISTORY.TXT」という名前のファイルを作成します。このファイルには、SAVE2FILE の値に応じた数の解が保存されます。たとえば、SAVE2FILE = 1000 に設定した場合、MIDACO は最新の解 1000 個の履歴をこのファイルに保存します。すべての解を保存したい場合は、SAVE2FILE を大きな値に設定してください。たとえば、SAVE2FILE = 10000000 に設定すると、MIDACO は履歴ファイルに最大 1 千万個の解を保存します。

解履歴ファイルの作成は、CPU 時間を大量に消費する事例で、利用できるすべての評価結果をもれなく入手したい場合に役立ちます。特に並列化を行う場合は、解履歴ファイルを作成すれば、処理されたすべての評価を追跡できます。
解履歴ファイルで使用される解の形式は、多目的問題用に作成されたパレートフロントファイル「MIDACO_PARETOFRONT.TXT」で使用される形式と同じなので、解履歴ファイルの内容も PlotTool でプロットすることが可能です（セクション6参照）。

重要：**並列化** した場合、SAVE2FILE の値は、個々の評価ではなく、ブロック *blocks* の数を数えます。たとえば、P=30、SAVE2FILE=10 の場合、10 個の解答ではなく、300（=30×10）個までの解が解履歴ファイルに保存されます。多くの解が保存されると、解履歴ファイルのサイズが**非常に大きくなる** ので注意が必要です。

3 MIDACO の停止基準

MIDACO の停止基準は、「ハードリミット基準」と「アルゴリズム基準」の 2 グループに分類されます。ハードリミット基準である MAXTIME と MAXEVAL は、CPU 時間の最大割り当てや関数評価の回数に基づいて MIDACO の最適化プロセスを停止させます。アルゴリズム基準である FSTOP・ALGOSTOP・EVALSTOP は、アルゴリズム上の決定に基づいて MIDACO を停止させます。すべての停止基準は、ユーザーが目的次第で自由に組み合わせ可能です。以下のサブセクションでは、各基準の詳細を説明し、いくつかの設定例を示します。

3.1 ハードリミット基準

以下では、時間および評価回数というハードリミットに基づく停止基準について説明します。

3.1.1 MAXTIME

MAXTIME 基準は、最大の CPU 時間割り当てを秒単位で定義します。この停止基準は自由に設定可能です。非常に大きな値 (1000000 など) を設定すると、この基準は実質的に無効になります。MIDACO が提供するほとんどのサンプル問題では、60*60*24（=1 日）のダミー値を使用しています。以下の表で、一般に基準とされる時間について、参考までに秒単位で表示します。

1 分	:	60	=	60
15 分	:	60*15	$<\approx$	1000
1 時間	:	60*60	=	3600
2.5 時間	:	60*60*2.5	$<\approx$	10000
1 日	:	60*60*24	=	86400
27 時間	:	60*60*27	$<\approx$	100000
1 週間	:	60*60*24*7	=	604800

3.1.2 MAXEVAL

MAXEVAL 基準は、問題関数の評価の最大割り当て回数を定義します。MIDACO ソルバーを組み込む際の優れた特徴として、任意の評価回数 (たとえば 123456) で正確に停止できることが挙げられます。つまり、ユーザーは MAXEVAL に任意の整数値を自由に選ぶことが可能です。

MIDACO は、実際の関数評価が（ベンチマーク問題のように）計算量の少ないものであれば、何百万もの関数評価を数秒で処理できます。高速で計算を行う事例では、数分以内に評価の限界である 10000000（1 千万）または 100000000（1 億）に達することが多くあります。このように計算の速い事例では、MAXEVAL の停止基準を完全に切り替えることが望ましい場合があります。その際、MAXEVAL 基準は 999999999（「9 を 9 個 *nine times nine* 」）よりも小さい値にのみ適用されます。MAXEVAL の停止条件に 999999999 やそれ以上の値が割り当てられている場合、MAXEVAL 条件は完全に無効となります。

なお、上記の特別なシナリオとは対照的に、数千または数百の評価を合理的な時間内に計算することができないような、CPU 時間のかかる 利用事例の特殊なケースについては、特にセクション7で扱っています。

3.2 アルゴリズム基準

以下では、アルゴリズムの判断 に基づいた停止基準について説明します。

3.2.1 FSTOP

FSTOP パラメータは、厳密なゼロ（0.0E+0）以外の値が割り当てられると有効になります。MIDACO が FSTOP 以下の目的関数値で実行可能な解に達した場合、MIDACO は停止します。この停止基準は、多目的問題の場合、最初の目的関数を参照することにご注意ください。MIDACO は FSTOP の値を厳密に扱うため、MIDACO が FSTOP の値に許容範囲をつけ加えない点が重要です。FSTOP の値としてゼロが望ましい場合は、0.000000001 などの極小値を FSTOP の値として使用できます。

3.2.2 ALGOSTOP

ALGOSTOP パラメータは、任意の正の整数値 (たとえば１・２・３...) が割り当てられると有効になります。この基準は、MIDACO 内部での ACO 再起動時のアルゴリズムの改善を測定します。ALGOSTOP の値は、(実行可能な) 目的関数値が改善されない場合の、MIDACO 内部の ACO の連続した再起動の最大数を定義します。たとえば、ALGOSTOP=10 が設定されている場合、MIDACO は現在の解が改善されなくなるまで、10 回の連続した内部 ACO の再起動 で最適化探索を実行します。

ALGOSTOP の値が高くなるほど（たとえば 10・50・100・それ以上)、MIDACO が全体最適解に到達する可能性が高くなります。このような観点から、この停止基準は大域的な最適性を示す最も進んだ？advanced？基準です。この停止基準の重大な難点は、多くの (通常は数千または数百万の) 関数評価が必要になることです。そのため、評価にかかる CPU 時間が少ない事例にのみ適しています。ALGOSTOP 基準を試す際には、特定の事例における実行時間効果を実感できるように、まずは 1・5・10・30 などの値を使用するとよいでしょう。

CPU 時間のかかる評価を行う事例に対しては、EVALSTOP 基準を用いる方が適しています。

3.2.3 EVALSTOP

EVALSTOP パラメータは、正の整数値（1・2・3... など）が割り当てられると有効になります。ALGOSTOP 基準と同様に動作しますが、大きく異なるのは、EVALSTOP 基準が MIDACO 内部の ACO の完全な再起動を考慮せず、個々の関数の評価を考慮する点です。たとえば、EVALSTOP=999 が設定されている場合、MIDACO は、暫定解が連続 999 回の関数評価 で改善されなくなるまで、最適化探索を実行します。

EVALSTOP の値を低く設定すると（たとえば、1000・100・それ以下)、MIDACO はより短時間で停止します。EVALSTOP=1 が設定されている場合、MIDACO は、暫定解を改善しない関数評価の後、直ちに停止します。したがって、ごく小さな EVALSTOP 値（1・2・3...）では、MIDACO は非常に短時間で停止します。この停止基準の目的は、必要な関数評価の数という点で、ALGOSTOP ほど多くを必要としないながら

も、アルゴリズムの尺度に基づく停止基準を提供することです。EVALSTOP 基準を試す場合、特定の事例で実行する効果を知るために、最初は 10000・1000・500 などの値を使うとよいでしょう。

EVALSTOP 基準は、新たな解が改善されたかどうかを測定するための精度（相対的なパーセンテージ）を指定することで、さらに細かく調整できます。EVALSTOP のデフォルトの精度 0.001 で、これは相対的なパーセンテージでは 0.1% です。異なる精度を使用する場合は、その精度を、EVALSTOP 値に浮動小数点として追加する必要があります。たとえば、MIDACO が 333 回連続した関数評価を実行し、目的関数値が相対的に 0.25% 改善されない場合は停止する、という停止条件は、EVALSTOP= 333.0025 と設定することで有効になります。EVALSTOP に特定の浮動小数点の数値が付与されていない場合、自動的にデフォルト精度 0.001 が適用されます。

3.3 サンプル事例

ここでは、単一または複数の停止条件を同時に設定する方法について、いくつかのサンプル事例を紹介します。

3.3.1 1 回のみの評価

MAXEVAL=1 と設定されている場合、MIDACO は関数評価を 1 回しか実行しません。これは、ユーザーが提供する開始点 X を定義しています。こうした設定は、問題の実装を検証するのに便利です。というのも、ユーザーは、開始点 X について報告されるすべての目的関数と制約関数の値が、妥当であるかどうかを詳細にチェックする機会を得られるからです。このオプションは、与えられた解を再現する (つまり再評価する) のにも便利です。

3.3.2 CPU 時間を要する事例

ここで紹介するのは、少ない数の関数評価しか利用できない、CPU 時間のかかる事例です（たとえば、複雑な機械シミュレーションモデルなど)。こうした事例には、以下のような設定がよいでしょう。

MAXTIME	=	50000
MAXEVAL	=	999999999 (→ 無効)
FSTOP	=	0 (→ 無効)
ALGOSTOP	=	0 (→ 無効)
EVALSTOP	=	50

この設定では、50000 秒（およそ半日）のハードなタイムリミットに加えて、EVALSTOP=50 の停止基準を設定し、タイムリミットに到達する前に EVALSTOP の停止基準に到達することを期待しています。他の基準はすべて無効になっています。

3.3.3 CPU 時間を要しない事例

次に、(学術的なベンチマーク問題のように）大量の関数評価を迅速に計算する、CPU 時間のかからない事例を見ていきます。こうした事例には、次のような設定が適しています。

MAXTIME	=	60*60*24
MAXEVAL	=	10000000
FSTOP	=	0.00000001
ALGOSTOP	=	200
EVALSTOP	=	0 (→ 無効)

この事例では、ハードリミットである関数評価回数基準を 100000000（1 千万）回に設定しており、さらに、FSTOP=0.00000001 および ALGOSTOP=200 の基準を設定しています。したがって、MIDACO が停止するのは、目標値が 0.00000001 以下の（実行可能な）解が見つかった場合、MIDACO 内部の ACO が 200 回連続して再起動しても解が改善されない場合、あるいは評価の割り当て回数が費やされた場合のいずれかです。この事例では実際の時間を気にしないので、MAXTIME を丸一日とすることで実質的に無効にしています。

3.3.4 無限実行

MIDACO のソフトウェアは、テキストファイルの出力を除けば、いかなるデータやメモリも蓄積せず、内部のアルゴリズム要素が何らかの制限でクラッシュすることもないため、機能的には無限に動作できるよう構築されています。

FSTOP、ALGOSTOP、EVALSTOP をデフォルト値 (ゼロ) のままにして、MAXEVAL と MAXTIME を 999999999 （「9 を 9 個 *nine times nine* 」）のような大きな値に設定すると、MIDACO が実質的に無限に動き続けるような設定が可能になります。このような設定は、一見して不合理に思われるかもしれませんが、実際はよく使われています。MIDACO のすべての自動停止基準を無効にすることで、ユーザーは最適化の実行をいつ停止するか（たとえば、プログラムやコンピュータをシャットダウンするなど）を自分で決めることができ、MIDACO が大域的最適解（あるいはパレート点の最適な広がり）を見つけるチャンスがかなり高まります。

注目いただきたいのは、MIDACO_SOLUTION.TXT と MIDACO_PARETOFRONT.TXT ファイルが、PRINTEVAL 関数の評価頻度ごとに、最新の解に上書きされることです。ユーザーは、無限実行の途中でも解にいつでもアクセスできることに加え、MIDACO 最適化が実行中であっても、解をさらに処理したり、パレートフロントをプロットしたりすることができます。

4 MIDACO のパラメータ

MIDACO は、性能や動作をカスタマイズする 13 種類のパラメータを備えています。以下のサブセクションで、個々のパラメータを説明します。なお、すべてのパラメータのデフォルト値は、ゼロです。

4.1 PARAM(1) : ACCURACY

このパラメータでは、制約条件違反の精度の許容範囲を定義します。|G(X)| ≤ PARAM(1) の場合、MIDACO は、等式制約が実行可能であるとみなします。G(X) ≥ -PARAM(1) の場合、不等式制約が実行可能とみなされます。ユーザーが PARAM(1) = 0 と設定した場合、MIDACO はデフォルトの精度 0.001 を使用します。このパラメータは、制約付き問題を扱う MIDACO の性能に大きな影響を与えます。制約条件が多い、あるいは難しい問題では、あまり正確でない精度 (例:PARAM(1)=0.1 または PARAM(1)=0.05) を使ったいくつかのテスト実行から始め、その後、より高い精度 (例:PARAM(1)=0.0001 または PARAM(1)=0.0000001) を使ったいくつかの洗練された実行を適用することをお勧めします。

なお、表示「VIOLATION OF G(X)」(MIDACO 画面参照) は、PARAM(1) に対するベクトル G の L_1 ノルムを表しています。すべての制約条件が PARAM(1) で定義された精度で実現可能な場合、「VIOLATION OF G(X) 」は 0 と表示されます。

4.2 PARAM(2) : SEED

このパラメータは、MIDACO の内部擬似乱数ジェネレータの初期シードを定義します。シードは、ジェネレーターによってサンプリングされる擬似乱数のシーケンスを決定します。したがって、この値を変更すると、MIDACO の結果が変わります。シードは、ゼロ以上の整数でなければなりません (たとえば、PARAM(2) = 0・1・2・3・...・1000 など)。

MIDACO は、同じシードを使う場合、同じコンパイラ設定で同じマシンによりプログラムを実行すれば、再現性があります。ただし、ハードウェア (CPU) やソフトウェア (コンパイラのバージョンやコンパイルフラグなど) を変更すると、結果が変わってしまう可能性があります。これは、内部の乱数ジェネレータが非常に敏感な性質を持っているためです。ユーザーが乱数シードを指定する主な利点は、期待できる実行を再現できることです。たとえば、プログラムの実行が意図せず中断され、実行を再開する必要がある場合などに有効です。もう一つの利点は、デバッグに利用できることです。

シードの影響は、問題の複雑さによって異なります。一般的には、問題が複雑であればあるほど、シードの影響は大きくなります。したがって、難しい問題の場合は、あまり長い時間をかけて MIDACO を実行するよりも、異なる乱数シードを使って、短い時間で MIDACO を実行する方がよい結果を期待できます。

4.3 PARAM(3) : FSTOP

このパラメータは、MIDACO の停止基準を有効にします。FSTOP の停止基準は、到達すべき目的関数の値に基づいています。FSTOP パラメータの詳細については、セクション 3.2.1 で説明しています。多目的問題の場合、FSTOP の値は最初の目的関数に適用されます。

4.4 PARAM(4) : ALGOSTOP

このパラメータは、MIDACO の停止基準を有効にします。ALGOSTOP の停止基準は、MIDACO のアルゴリズム・プロセスに基づいています。ALGOSTOP パラメータの詳細については、セクション 3.2.2 で説明しています。

4.5 PARAM(5) : EVALSTOP

このパラメータは、MIDACO の停止基準を有効にします。EVALSTOP 停止基準は、関数の評価回数を考慮した MIDACO のアルゴリズム・プロセスに基づいています。EVALSTOP パラメータの詳細については、セクション 3.2.3 で説明しています。

4.6 PARAM(6) : FOCUS

このパラメータは、MIDACO に現時点の最良解を中心とする探索プロセスをより集中的かつローカルに実行させます。これは**最も強力なパラメータのひとつ**で、幅広く適用できます。多くの問題で、このパラメータを調整することは有用であり、(特に凸問題や半凸問題では) 収束速度が速くなります。また、このパラメータは、特に暫定解を絞り込むのに役立ちます。PARAM(6) がゼロでない場合、MIDACO は、ガウス PDF の標準偏差の上限を適用します（「はじめに」図1を参照)。連続変数の標準偏差の上限値は、(XU(i)-XL(i))/FOCUS、整数変数の標準偏差の上限値は、MAX((XU(i)-XL(i))/FOCUS,1/SQRT(FOCUS)) で与えられます。

つまり、
FOCUS の値が大きければ大きいほど、MIDACO は現時点の最良解により近づいて検索を実行します。

PARAM(6) の値は整数でなければなりません。小さい値の FOCUS（たとえば 10 や 100）は最初のテストラン（特定の開始点なし）で推奨されます。FOCUS の値を大きくすると（たとえば 10000 や 100000)、通常は（特定の解を出発点とする）絞り込み実行の場合にのみ有効です。
さらに、FOCUS にマイナスの値を入力することも可能です（たとえば-1000 や-10000)。この場合 MIDACO は、マイナス (「-」) を数値としては扱わずに、情報フラグと見なします。正の FOCUS 値の場合、MIDACO は独立した再起動を行って探索空間の他の領域も探索しますが、負の FOCUS 値の場合は、MIDACO 内の独立した再起動オプションを無効にします 。言い換えれば、負の FOCUS 値の場合、MIDACO は開始点のみに集中します。したがって、負の FOCUS 値は、開始点として使用された特定の解の質に高い信頼性がある場合の絞り込み実行に限定して使用してください。

4.7 PARAM(7) : ANTS

このパラメータでは、MIDACO が 1 つの世代（進化的 ACO アルゴリズムの主要な反復）の中で生成する蟻 *ants*（反復子）の数を固定することができます。このパラメータは、PARAM(8) と組み合わせて使用します。ANTS と KERNEL のパラメータの使用は、いくつかの問題（特に、大規模な問題や CPU 時間を多用する事例）で降下を期待できます。ただし、これらのパラメータを調整すると、MIDACO のパフォーマンスが著しく低下する可能性があります。PARAM(7) がゼロの場合、MIDACO は世代ごとの蟻の数を動的に変化させます。このパラメータの取り扱いについては、PARAM(8) を参照してください。

4.8 PARAM(8) : KERNEL

このパラメータでは、MIDACO のマルチカーネル・ガウス PDF のカーネル数を固定することができます（「はじめに」図 1 参照）。カーネルの数は、MIDACO の暫定解アーカイブに保存されている解の数にも対応します。凸型傾向の問題では、カーネル数が少ないと収束が早くなり、カーネル数が多いと収束が悪くなることがわかります。他方で、カーネル数が小さいと、MIDACO が局所最適解に陥る危険性が高くなり、カーネル数が大きいと、大域最適解に到達する可能性が高くなります。
KERNEL パラメータは、ANTS パラメータとの組み合わせで使用する必要があります。以下の表2に、ANTS と KERNEL を組み合わせた設定例を示します。

表2: ANTS/KERNEL の設定パターン例

パターン 1		パターン 2		パターン 3		パターン 4	
ANTS	2	ANTS	30	ANTS	500	ANTS	100
KERNEL	2	KERNEL	5	KERNEL	10	KERNEL	50

パターン 1 は、最小の設定です。この設定は、数百の関数評価しかできないような、非常に CPU 時間のかかる問題や、特定の（たとえば凸性などの）構造を持つ問題では有効かもしれません。2 つめの設定は、比較的少ない数の ANTS しか考慮しないので、CPU 時間のかかる問題にも使用されるかもしれません。3 番目と 4 番目の設定は、評価時間が速い問題でのみ効果を期待できます。ANTS と KERNEL のパラメータの調整は問題に大きく依存するため、ユーザーはこうした値を試す必要があります。

4.9 PARAM(9) : ORACLE

このパラメータでは、MIDACO 内のペナルティ関数に対するユーザー指定のオラクルパラメータを指定します。このパラメータが関係するのは、制約付き問題のみです。PARAM(5) がゼロでない場合、MIDACO は PARAM(5) を初期オラクルとして使用します（ゼロの場合、MIDACO は 10^9 を初期オラクルとして使用します）。
このオプションは、問題の背景知識が存在する制約付き問題では特に有効です。たとえば、ある事例で、F(X)=1000 に対応する実現可能な解 X （たとえば、プラント運転コストの US ドル金額）を持っていることがわかっているとします。800 や 600 のオラクル値を MIDACO に与えることは、このコスト領域に新しい実

行可能解がある（そのコスト値でプラントを運転できる）かもしれないため、合理的かもしれません。他方で、1000 以上のオラクル値はユーザーには関心がなく、低すぎる値（たとえば 200）は合理的ではありません。オラクルペナルティ法の詳細については、オラクルペナルティ法のウェブサイト をご覧ください。

4.10 PARAM(10) : PARETOMAX

このパラメータは、MIDACO が保存する非支配解（パレート点 *pareto points* とも呼ばれる）の最大数を定義します。MIDACO が使用するデフォルト値（PARAM(10)=0 が設定されている場合）は 1000 です。PARETOMAX には、PARAM(10)=333 や PARAM(10)=100000 など、任意の大きな整数を自由に設定することができます。PARETOMAX の値を大きくすると、より多くのメモリを必要とし、パレート優劣判定のフィルタリング作業が増えるため、通常は、MIDACO 内部の計算時間が遅くなることにご注意ください。多くの事例では、PARETOMAX 値 <= 1000 で十分です。PARAM(10) = 30 のように小さな値を指定することも可能で、その場合は通常、MIDACO 内部の計算時間が速くなります。

4.11 PARAM(11) : EPSILON

このパラメータは、MIDACO が多目的パレート優劣判定フィルタに使用する精度を定義します。デフォルトの PARAM(11)=0 が設定されている場合、2 つの目的関数を持つ問題には EPSILON=0.001 の値が、3 つ以上の目的関数を持つ問題には EPSILON=0.01 の値が使用されます。EPSILON の値が低いほど、パレートフロントに新しい解が導入される可能性が高くなります。したがって、EPSILON の値は、MIDACO が保存するパレート点の量と内部計算時間に、大きな影響を与えます。

大半の事例では、EPSILON の値は 0.001 以上で十分です。0.00001 や 0.00000001 といった小さな値では、ふつう（きわめて）多くのパレート点が報告されます。しかし、それらのパレート点は互いにわずかな違いしかなく、あまり有用な情報を提供しないかもしれません。多目的最適化の特殊なケースとして、4 つ以上の目的関数を考慮する多数目的問題があります。これらの問題では、大きな数（数千など）の非支配的な解がただちに生成されます。多数目的問題では、PARAM(11)=0.01 や PARAM(11)=0.1 のように、より大きな EPSILON 値を割り当てることが有用です。このような大きな EPSILON 値を使うと、MIDACO は少なくとも一つの目的関数につき、大きな差があるパレート点だけを保存して報告するようになります。大きな EPSILON 値を使うと、MIDACO 内部の計算時間が大幅に短縮されることにも注目してください。

4.12 PARAM(12) : BALANCE

BALANCE パラメータは、多目的問題に関して大きな影響を及ぼします。BALANCE は、パレートフロントのどの部分／領域に MIDACO の探索努力を主に集中させるかを定義します。デフォルトでは、MIDACO は、すべての目的関数の間で最もバランスのとれたトレードオフを提供するパレートフロントの部分に、探索努力のほぼすべてを集中させます。BALANCE パラメータを使用すると、パレートフロントのどの部分にも焦点を当てることができます。このパラメータの詳しい説明と事例は、セクション 5.2 を参照してください。

4.13 PARAM(13) : CHARACTER

CHARACTER パラメータでは、MIDACO 内部の パラメータ設定を有効にすることができます。MIDACO では、以下の 3 種類があらかじめ定義されています。

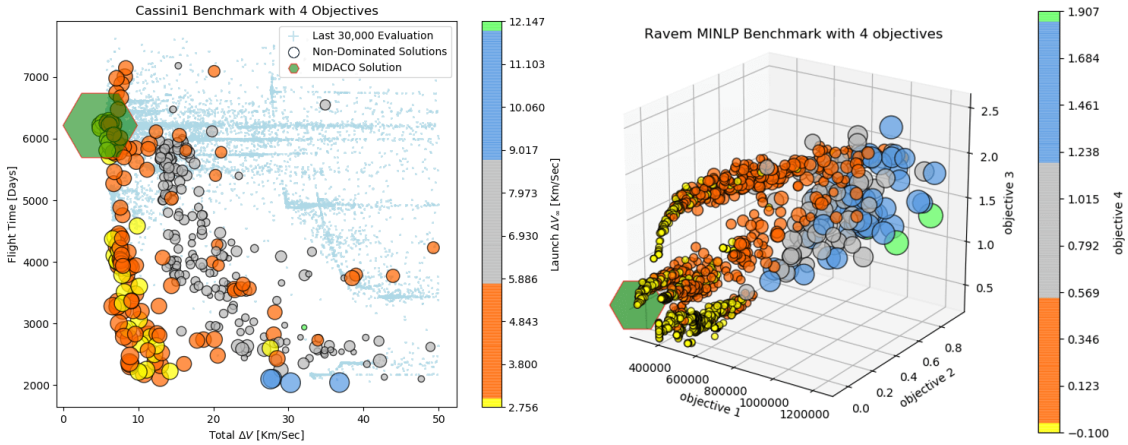
CHARACTER = 1	:	連続関数問題タイプの MIDACO 内部パラメータ
CHARACTER = 2	:	組み合わせ関数問題タイプの MIDACO 内部パラメータ
CHARACTER = 3	:	*All-Different* 問題タイプの MIDACO 内部パラメータ

PARAM(13)=0 が設定されていると、MIDACO は、連続関数問題タイプの内部パラメータを選択するか、組み合わせ関数問題タイプの内部パラメータを選択するかを自分で決定します。連続関数問題タイプの内部パラメータを選択すると、よりきめ細かい探索処理が可能になり、組合せ関数問題タイプの内部パラメータを選択すると、より大雑把な探索処理が可能になります。

特殊なケースとして、*All-Different* 問題タイプがあります。これらの問題では、すべての整数変数に異なる値が含まれていなければなりません。*All-Different* 問題の有名な例は、巡回セールスマン問題 (TSP) です。*All-Different* 問題の場合は、CHARACTER=3 を有効にする必要があります。CHARACTER=3 が設定されていると、MIDACO は自動的に *All-Different* 条件を満たす解のみを生成します。つまり、*All-Different* 条件を明示的に定式化して、制約条件ベクトル G(X) として 提示する必要はなく、MIDACO が自動的に処理するのです。*All-Different* 設定を使用する場合は、開始点 X がすでに *All-Different* 条件を満たす必要があることに注意してください、そうでない場合は IFLAG=402 エラーが発生します。

なお、MIDACO の *All-Different* 設定は、混合整数問題にも使用できます。その場合、*All-Different* 条件はすべての整数変数に影響を与えますが、連続変数には影響しません。

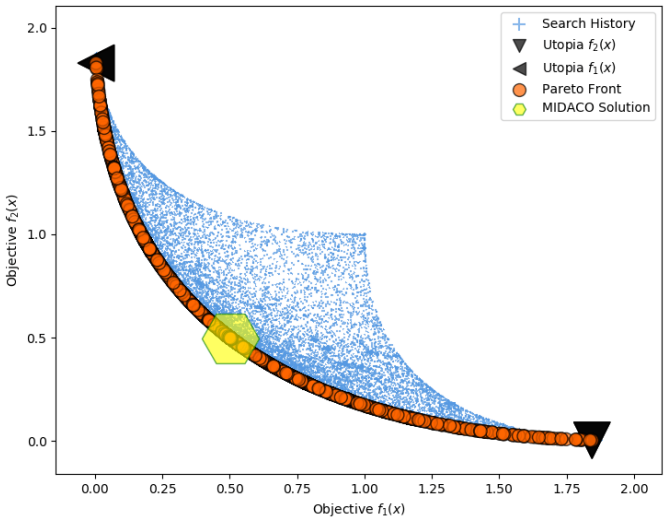
5 多目的最適化



多目的最適化では、複数の目的関数を同時に扱います。大域的最適解として（通常は）単一の解が存在するような、単一の目的関数の最適化とは対照的に、多目的最適化では、それぞれの目的関数を同時に最適化する単一の解は（通常は）存在しません。その代わり、個々の目的関数の間の**トレードオフ曲線**（パレートフロント Pareto Front ともいう）を表す、非支配的な解（パレート最適 *pareto-optimal* ともいう）のセットが存在します。たとえば、２つの目的関数 $f_1(x)$ と $f_2(x)$ がある、次のような多目的おもちゃ問題を考えてみましょう。

$$\text{Minimize} \quad \begin{cases} f_1(x) = (x_1 - 0)^2 + (x_2 - 0)^2 \\ f_2(x) = (x_1 - 1)^2 + (x_2 - 1)^2 \end{cases} \qquad \text{with} \qquad x_1, x_2 \in [0, 1]$$

そして、このおもちゃ問題におけるパレートフロントは次の図のようになります。



MIDACO を使った多目的最適化は、簡単です。ユーザーは、問題関数の次元宣言（すべてのサンプルファイルの *STEP 1.A*）で、対応する MIDACO 入力パラメータを使って目的関数の数を示すだけです。上のおもちゃ問題では、次のようになります。

```
% STEP 1.A: Problem dimensions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
problem.o  = 2; % Number of objectives
problem.n  = 2; % Number of variables (in total)
problem.ni = 0; % Number of integer variables (0 <= ni <= n)
problem.m  = 0; % Number of constraints (in total)
problem.me = 0; % Number of equality constraints (0 <= me <= m)
```

MIDACO は、**パレートフロント全体**を提示することで多目的問題を完全に自動で解決し、特にパレートフロントの 1 箇所を *MIDACO Solution* として強調表示します。上記のおもちゃ問題を解決するための MIDACO コンソール画面は次のようになります。

```
----------------------------------------
| OBJECTIVES    2 | PARALLEL         1 |
|--------------------------------------|
| N             2 | MAXEVAL     100000 |
| NI            0 | MAXTIME      86400 |
| M             0 | PRINTEVAL    10000 |
| ME            0 | SAVE2FILE        1 |
|--------------------------------------|
| PARAMETER:    All by default (0)     |
----------------------------------------

[     EVAL,    TIME]  MULTI-OBJECTIVE PROGRESS  VIOLATION OF G(X)
-------------------------------------------------------------------   [PARETO]
[        1,       0]   PRO:          2.00000000   VIO:     0.000000   [     1]
[    10000,       0]   PRO:        -13.52562586   VIO:     0.000000   [   286]
[    20000,       0]   PRO:        -16.00900246   VIO:     0.000000   [   327]
[    30000,       0]   PRO:        -16.78937764   VIO:     0.000000   [   361]
[    40000,       0]   PRO:        -16.78937764   VIO:     0.000000   [   398]
[    50000,       1]   PRO:        -20.30755765   VIO:     0.000000   [   472]
[    60000,       1]   PRO:        -20.30755765   VIO:     0.000000   [   529]
[    70000,       1]   PRO:        -22.59627173   VIO:     0.000000   [   551]
[    80000,       1]   PRO:        -25.16010495   VIO:     0.000000   [   574]
[    90000,       1]   PRO:        -25.16010495   VIO:     0.000000   [   591]
[   100000,       1]   PRO:        -26.78532572   VIO:     0.000000   [   597]

OPTIMIZATION FINISHED  --->  MAXEVAL REACHED


        BEST SOLUTION FOUND BY MIDACO
-------------------------------------------
EVAL:    100000,  TIME:    1.00,  IFLAG:   1
-------------------------------------------
PROGRESS                -26.785325716864353
-------------------------------------------
NUMBER OF PARETO POINTS                 597
-------------------------------------------
f[   0] =                0.495602836738631
f[   1] =                0.504550856899625
-------------------------------------------
x[   0] =               0.491982059221132;  # __________x__________
x[   1] =               0.503543930698371;  # __________x__________
```

Current number of non-domiated solutions, also called *pareto-points*

5.1 Multi-Objective Progress（PRO）関数

多目的問題を解く場合、MIDACO は計算画面出力のメインカラムに *Multi-Objective Progress (PRO)* 関数の値を表示します。この機能は、MIDACO の**ユニークな機能** で、多目的最適化プロセス全体をモニターするための測定値として機能します。この関数の表示と動作は、単目的最適化関数と意図的に似せており、単目的問題から多目的問題への移行を可能な限りスムーズにします。

MIDACO は、一般性を失うことなく、すべての目的関数を最小化するものと考えます。そのため、Multi-Objective Progress 関数の値が小さいほど、ポジティブな進捗を意味します。BALANCE パラメータのデフォルト値（ゼロ）や 1 より小さい値を使用する場合、Multi-Objective Progress 関数の値は MIDACO 内部でのみ意味を持ち、問題の目的関数の値とは直接関係しません。PRO 値を表示する目的は、ユーザーに進捗状況をフィードバックすること にあります。PRO 値が改善された（値が下がった）ということは、パレートフロント全体で何らかの改善があったことを意味します。通常、これはより多くの、あるいはより良いパレート点が発見されたことを意味します。また、（パレートフロントの 1 点である）MIDACO の解が直接改善されたことも示します。このような改善は、いくつかの目的関数値を下げること、および／または、パレートフロントの中で MIDACO の解を再配置することを意味します。

PRO 機能の正確な計算は複雑であり、特に**多数目的**最適化問題について [43] で採用されたユートピア・ネイディア・バランス *Utopia-Nadir-Balance* のコンセプトに基づいています。

5.1.1 BALANCE パラメータを１つの目的関数のみに設定

特殊なケースとして、BALANCE パラメータ（下のセクション5.2を参照）が複数の目的関数の 1 つだけに設定される場合があります。その場合、表示される PRO 値は、BALANCE パラメータが割り当てられている目的関数の値と**同一** になります。これにより、ユーザーは PRO 値を介して選択された目的関数の進捗状況を直接モニターすることができます。**この手法が多くの場合で効果を発揮する** のは、複数の目的関数のうちの 1 つが他の目的関数よりも解くのがきわめて困難な場合や、ある 1 つの目的関数が他の目的関数よりも重要度がきわめて高い場合です。

5.2 BALANCE パラメータ

デフォルトの設定では、MIDACO は、すべての目的関数の中で、**最もバランスのとれた** 解を示すパレートフロントの点に探索を特に集中させます。パレートフロントの別の部分に探索を集中させたい場合は、BALANCE パラメータの設定により可能となります（セクション 4.12）.

複数の目的関数のうち、特定の目的関数だけに集中して探索を行う必要がある 場合には、BALANCE パラメータを目的関数のインデックス番号に等しく設定すれば、簡単に実行できます。たとえば、BALANCE = 1.0 が設定されている場合、MIDACO は最初の目的関数にのみ検索を集中させます（上記のセクション5.1.1を参照）。BALANCE = 2.0 で設定されている場合、MIDACO は探索を 2 番目の目的関数にのみ集中させ、以下同様に、３番目、４番目、５番目 ･･･ の目的関数に探索を集中させます。

目的関数の間で優先順位が異なっており、パレートフロントの特定の部分で探索を**微調整する**必要がある場合は、その情報を BALANCE パラメータの各 10 進数を介して引き渡すことができます。この場合、個々の目的関数に 0 から 9 の**重要度**（または**優先度**）の値を割り当て、その値を BALANCE パラメータの対応する小数部に配置する必要があります。
これは、一見すると複雑に見えるかもしれませんが、実際はとても簡単です。たとえば、2 つの目的関数を持つ問題で、2 番目の目的関数に 1 番目の目的関数の**2 倍の重要性**を与える場合を考えましょう。この場合、BALANCE パラメータ設定は、BALANCE = 0.12 または 0.24 または 0.36 または 0.48 となります。どの設定でも、BALANCE パラメータは、小数第 2 桁（2 番目の目的関数に対応）が、第 1 桁（1 番目の目的関数に対応) よりも 2 倍高くなるように設定されています。

別の例として、1 番目の目的関数が 2 番目の目的関数よりも**4 倍の重要性**を持つ場合を考えます。この場合の BALANCE パラメータ設定は、1 桁目（1 番目の目的関数の重要性）が 2 桁目（2 番目の目的関数の重要性）の 4 倍になるため、BALANCE = 0.41 または 0.82 となります。

最後に、2 番目の目的関数が 1 番目の目的関数に対して**9 倍の重要性**を持つ例を考えましょう。この場合、2 桁目（2 番目の目的関数の重要性）が 1 桁目（1 番目の目的関数の重要性) の 9 倍となるため、パラメータ設定は BALANCE = 0.19 となります。下の図 3 は、先に検討したおもちゃ問題のパレートフロントについて、BALANCE パラメータを変化させた際の MIDACO の解の位置への影響を示したものです。

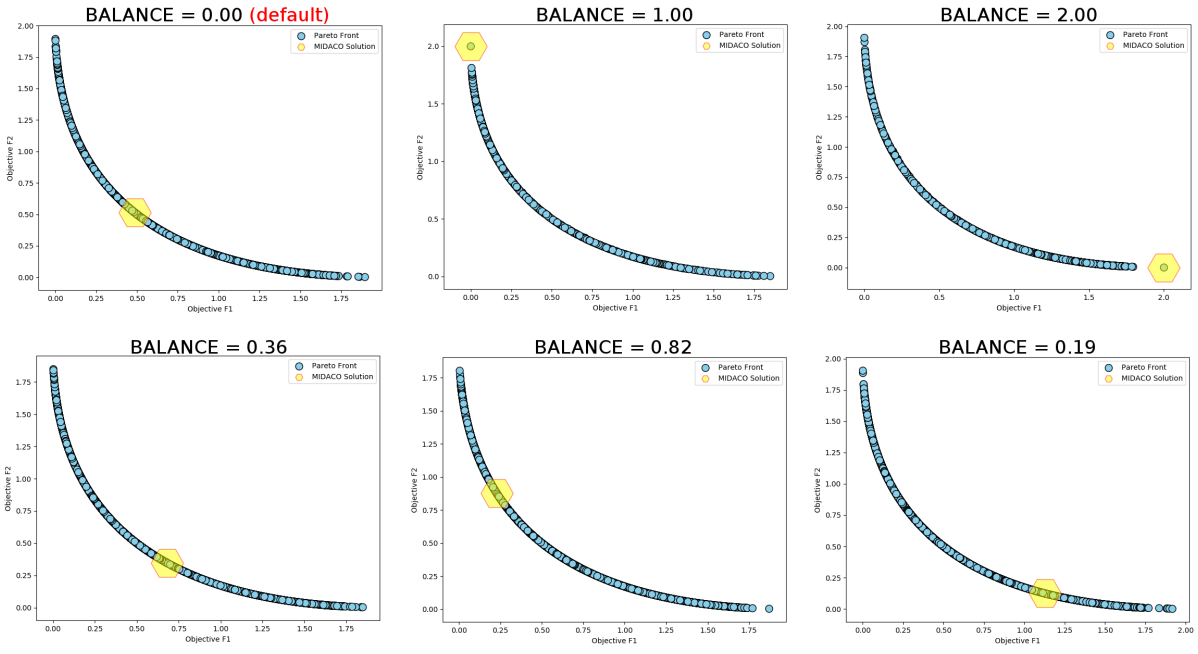
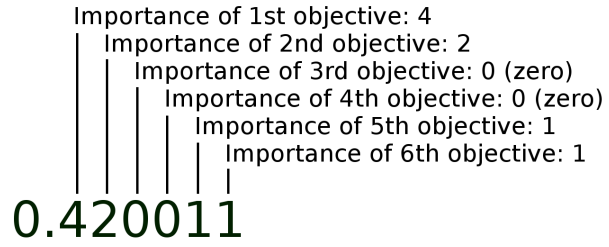


図3　BALANCE パラメータがパレートフロント上の MIDACO 解に対して与える影響

なお、数値処理上の理由から、BALANCE パラメータの微調整は、最初の 8 つの目的関数に対してのみ有効であることにご注意ください。BALANCE パラメータ設定が MIDACO に引き渡される場合、第 8 桁の位置より下の各桁は自動的に**重要度ゼロ** として扱われます。さらに留意いただきたいのは、重要度ゼロはどの目的関数にも割り当てられることです。この機能は、多くの目的関数を持つ問題で、いくつかの目的関数をモニターするだけでよい（ただし、探索の対象からは除外する）場合に有効です。たとえば、最初の目的関数が最も重要で、2 番目はその半分、3 番目と 4 番目の目的関数は重要度ゼロ、5 番目と 6 番目の目的関数は 2 番目の目的関数の半分の重要度である、 6 つの目的関数を持つ問題を考えてみましょう。この場合、BALANCE = 0.420011 と設定することで、すべての情報をひとつの数値として MIDACO に引き渡すことができます。



BALANCE パラメータの微調整は、**多数目的** 最適化問題の場合に特に有効です。多数目的問題は解くのが難しいことで知られていますが、パレートフロントの特定の部分に注目するほうが、パレートフロント全体で最適な収束を得ようとするよりも効果的な場合があります。

重要： 数値処理上の理由から、BALANCE パラメータを設定する場合は、**値の末尾にゼロ**を付けることをお勧めします。たとえば、BALANCE 値「0.15」をそのまま MIDACO に渡すのではなく、「0.15000000」という値を使ってください。その理由は、プログラミング言語によって（C++ や Fortran など）、最後の桁が二重表現に置き換えられてしまうからです。上の例の「0.15」の場合、「0.14999999」と処理されて、MIDACO に誤った情報を伝えることになり、結果が大きく変わってしまう可能性があります。

5.2.1 完全フロント探索の無効化

BALANCE パラメータをさらに微調整して、数値にマイナスの「-」フラグを追加することもできます。たとえば、BALANCE = 0.12000000 の代わりに、BALANCE = -0.12000000 を設定すると、MIDACO は内部の完全フロント探索のヒューリスティックをすべて無効にするため、パレートフロントの任意の特定部分にフォーカスを若干絞る ことができます。

複数の目的関数のうち、1 つの目的関数だけに集中する場合、マイナスの「-」フラグを設定することも可能です。たとえば、1 番目の目的関数だけに集中する場合は、BALANCE = 1.0 ではなく、BALANCE = -1.0 と設定します。ただし、BALANCE パラメータに負の「-」フラグを設定しても、その効果は一般的にはさほどなく、通常は何千回もの関数評価を行って初めて見えてくるものとなります。

5.3 パレートフロントのデータ

多目的最適化問題を解く際に、SAVE2FILE パラメータを >= 1 に設定した場合、MIDACO は自動的に **MIDACO_PARETOFRONT.TXT** という名前のテキストファイルを作成します。このファイルには、パレートフロント近似の全体像が含まれており、PRINTEVAL の頻度で作成されます（セクション2.1参照）。プロットツール（セクション 6 参照）を使うと、MIDACO_PARETOFRONT.TXT ファイルのデータをグラフで表示することができます。

ユーザーは、テキストファイルの代わりに、pf 配列を経由してパレートフロントのデータを得ることも可能です。pf 配列は、MIDACO のソースコード・ルーチン (上位言語の場合はライブラリ・ファイル) の入出力引数で、パレートフロント情報全体を格納し、上記のテキストファイルを作成するために使用されます。pf 配列の一番最初の要素には、パレート点の数が格納されています。ゼロインデックスで始まるプログラミング言語（たとえば Python）では、この情報は pf[0] で与えられます。1 インデックスで始まるプログラミング言語（たとえば R）では、この情報は pf[1] で与えられます。保存されているパレート点の数（*psize* という）がわかれば、以下のようにして個々の解にアクセスすることができます（擬似コード、1 インデックスから開始)。

```
psize = pf[1]  # pf[] 配列に格納されているパレート点の数

pfmax = 1000   # 最大パレート点のデフォルト値

for k=1:psize

      for i=1:o   # 目的関数

            f[i]  =  pf[ 2 + o*(k-1)+i-1 ]

      for i=1:m   # 制約条件

            g[i]  =  pf[ 2 + o*pfmax + m*(k-1)+i-1 ]

      for i=1:n   # 変数

            x[i]  =  pf[ 2 + o*pfmax + m*pfmax + n*(k-1)+i-1 ]
```

pf 配列を介してパレートフロントのデータに直接アクセスしたいユーザーは、それぞれのプログラミング言語や MIDACO ゲートウェイコードで指定されている「print_paretofront」サブルーチンコードを参照することもできます。「print_paretofront」サブルーチンは、上記の擬似コードを実行してパレートフロントのテキストファイルを作成し、目的に応じてコピーや編集することも可能です。

5.4 パレート点の数

パレート点の数は、PARETOMAX および EPSILON パラメータで設定できます。

デフォルトでは、MIDACO は最大で 1000 パレート点を収集します。この最大数は、PARETOMAX パラメータを任意の値に設定することで変更できます。PARETOMAX = 5000 と設定した場合、MIDACO は最大で 5000 のパレート点を保存します。また、PARETOMAX = 60 と設定した場合、MIDACO はパレート点を 60 までしか保存しません。

EPSILON パラメータ（セクション4.11を参照）は、ある解がパレートフロントに含まれるかどうかの決定に影響する許容精度を定義します。EPSILON の値が小さいほど、解が含まれる可能性が高くなります。したがって、EPSILON の値を小さくすると、通常、より多くのパレート点が収集されます。しかし、それらの解は互いに少ししか違わないかもしれません。

できるだけ多くのパレート点を収集する ためには、大きな PAERTOMAX 値と小さな EPSILON 値を組み合わせてください。たとえば、PARETOMAX = 10000 と EPSILON = 0.00001 は、通常、多くのパレート点を集めることができます。大量のパレート点を集める際の難点は、MIDACO 内部の実行時間が大幅に増加することです。もうひとつの難点は、それらのパレート点の多くはわずかしか違わないので、あまり洞察を得られないということです。

多目的最適化問題に対する MIDACO 内部の実行時間を短縮する ためには、PARETOMAX の値を小さくし、EPSILON の値を大きくします。たとえば、PARETOMAX = 100 と EPSILON = 0.005 で設定すると、MIDACO 内部の実行時間を大幅に短縮することができると同時に、十分に分散されたパレート点を得ることができます。多目的最適化問題で MIDACO 内部の実行時間を短縮することは、目的関数が多い問題では特に有益です。

5.4.1 多数目的最適化のための限定フィルタリング

多数目的最適化問題 では、BALANCE パラメータを調整し、一部の目的関数が重要度ゼロに設定されている場合 (セクション5.2を参照)、重要度ゼロの目的関数を、パレート優劣判定のフィルタリング作業から除外することができます。これは、PARETOMAX パラメータの数値に負の「-」フラグを追加することで可能です。たとえば、PARETOMAX = 500 ではなく PARETOMAX = -500 と設定した場合、MIDACO は最大 500 のパレート点を収集しますが、パレート優劣判定のフィルタリング基準は、BALANCE パラメータで示される正の重要性を持つ目的関数にのみ適用されます。

この機能を使用すると、パレート点の数を、正の重要性が割り当てられた目的関数についてのみパレート最適となるような **解のセットに減らす**ことができます。この機能は、多くの目的関数を持つ問題で、パレート点の数がすぐに大きくなって困るような場合に役立ちます。また、MIDACO 内部の実行時間も短縮できます。

6 プロットツール

プロットツール PlotTool は、**Windows**の実行プログラムで、Matplotlib [21] グラフィック・ライブラリに基づいています。PlotTool は、様々なソース（MIDACO 以外の最適化アルゴリズムも含む）からのパレートフロントのデータをグラフで図示するために利用できます。特に、MIDACO_PARETOFRONT.TXT と MIDACO_HISTORY.TXT ファイルのプロットに使用可能です。なお、**Linux**や**Mac**の場合、Wineを使って PlotTool.exe を実行することもできます。

> ウイルス対策ソフトによっては PlotTool.exe でウィルスが **誤って検知** される場合があります。

PlotTool.exe は、ソースファイルが置かれているのと同じフォルダーに保存することをお勧めします（が、必須ではありません）。実行ファイルをダブルクリックすると、プログラムが起動します。なお、Windows のセキュリティチェックのため、起動に時間がかかることがあります。PlotTool の主な機能の説明は省略し、以下ではいくつかの高度な機能について説明します。

6.1 値、色、カラーマップ、LaTeX サポート

ドロップダウンメニューで選択できる数値は、すべて自由に変更することができます。たとえば、*Marker Size*=123 や *Transparency*=0.456 などを入力可能です。これは、目的関数や制約条件・変数のインデックスが大きい場合、たとえば、目的関数 33、制約条件 44、変数 55 などの場合にも有効です。ドロップダウンで選択できる単色のほかにも、名前付きの色（シルバー、ゴールド、アクア、ライトピンク、ダークグリーンなど）や HEX カラーコード (#ff5733 など) が入力できます。名前付きの色の一覧はこちらをご覧ください。

https://matplotlib.org/examples/color/named_colors.html

Matplotlib がサポートするカラーマップはいずれも入力できますが、元のカラーマップ名の前に @ 記号を付けて入力します。たとえば、@viridis、@Reds、@winter、@summer、@Pastel1 などです。名前は小文字と大文字が区別されますのでご注意ください。名前の最後に「_r」を付ければ、どのカラーマップでも反転 させることができます (たとえば「rainbow_r」)。名前の付いた色の完全なリストは、以下を参照してください。

https://matplotlib.org/examples/color/colormaps_reference.html

タイトルと凡例のテキストは、LaTeX の一般的な構文コマンドに対応しています。
たとえば、「`$\alpha-\beta-\gamma$ Design`」というタイトルは、「$\alpha-\beta-\gamma$ Design」と表示されます。

6.2 追加データファイルと配置

メインのソースファイルに加えて、最大 3 つの追加ソースファイルを使用できます。各ファイルには、複数の解（解履歴ファイルなど）を含めることも、ひとつの解だけ を含めることもできます。プロットの配置順は、メインのソースファイルを前面に、最後に追加したファイルを背面に配置します。オプションボックスから、メインソースファイルのデータを背面に配置するオプション（Background Position) の選択も可能です。

6.3 解のエクスポートと再インポート

PlotTool の**重要な機能** は、ソースファイルから個々の解をエクスポートする機能です。２ D プロット上で、希望する解の点にマウスカーソルを置き、マウスの**右クリック**ボタンを押すと、ダイアログが表示され、それぞれの X 軸と Y 軸の座標で解をエクスポートするかどうかの確認を求められます。「はい（Y)」をクリックすると、PlotTool は、すべてのソースファイル（メインと追加分）の中から、マウスカーソルの位置に最も近い特定の解を探します。成功すると、新たにウインドウがポップアップして、解をテキストファイルに保存するよう求められます。このテキストファイルには、解の情報（F、G、X）がすべて含まれており、いくつかの一般的なプログラミング言語のフォーマットで表示されます。このテキストファイルから、解を精査し、コピーアンドペーストでさらに処理して、絞り込み実行の開始点 とすることができます。

この機能は、多数（数千）のパレート最適解を持つソースファイルでは特に有効です。ただし、大きなソースファイルでは、エクスポート検索処理に時間がかかる場合があります。

さらにこの機能は、プロット内の**特定のパレート点をハイライトする** 際にも便利です。一度テキストファイルにエクスポートされた解答は、「ADD FILE」オプションを使って簡単にプロットへの再インポートが可能です。追加のソースファイルについては、パレート点を区別するために、異なるサイズや、色・マーカーにすることができます。

6.4 保存・読み込み・リセット

PlotTool のユーザー設定全体を保存・読み込みすることができます。設定データを保存するファイルのデフォルト名は PlotTool.set です。このファイルは手動で変更することも可能で、GUI の位置やカラーバー・マークの数の変更 などの追加オプションが含まれます。変更内容は、同じフォルダーにある変更後の PlotTool.set ファイルを使って PlotTool を再起動すると有効になります。

「LOAD > Reset All to Default」で、すべての設定を**初期値にリセットする**ことができます。

6.5 Live モード

LIVE コマンドは、PlotTool がソースファイルの更新を常にチェックし、プロットにリアルタイムで表示します。LIVE コマンドは PLOT コマンドよりもエラー許容性に優れており、たとえば、まだソースファイルが存在しない 場合でも LIVE コマンドは実行されます。なお、LIVE モードは、ある程

度の CPU パワーを必要とするため、MIDACO による最適化の実行速度が（いくぶん）低下することにご注意ください。

警告: 場合によっては（VBA や MS-Visual など）、ライブモードで実行すると、MIDACO の最適化実行がクラッシュすることがあります。これは、PlotTool と MIDACO の両方がソースファイルにアクセスしようとして、ファイルアクセスの衝突が発生するためです。十分ご注意ください。

6.6 ズーム

2D プロットでは、マウスホイールを使ってズームイン／アウトすることができます。3D プロットでは、マウスの右ボタンを押したまま、マウスを前進または後退させることでズームが可能です。なお、解のエクスポートは 2D プロットでのみ可能です。

6.7 MIDACO カラーマップのカスタマイズ

プロットツールでは、通常の Matplotlib のカラーマップに加えて、とくに最適化の描画に利用しやすい 、カスタマイズ可能な MIDACO カラーマップを用意しています。このカラーマップは、５色（黄色、オレンジ、青、緑）と白で構成されており、色の範囲が外側になるほど指数関数的に小さくなるのが特徴です。最小（または最大）を示す一番外側が最も重要となるため、このカラーマップでは、この領域をより詳細に表示します。MIDACO のカラーマップには 2 つのパラメータがあります。それは、カラーマップ全体の中で、色が表示されるまでの割合 (%) と、色の変化に応じた指数関数 (ex) です。たとえば、「midaco_50%_ex2.5」というカラーマップは、カラーマップ全体の 50% を白のままにして、指数関数的な係数 2.5 に従って色を変化させます。指数係数は通常、2.0〜4.0 の間の浮動小数点数を選択します。以下の図は、Fonesca ベンチマークでカスタマイズされた MIDACO のカラーマップ４種類を示したものです。この図では、カラーマップの名前に「_r」を付けてさらに反転させています。さらに、より詳細な分析を行うために、カラーマップのマークの数を 10 から 30 に増やしていることにもご注目ください（セクション 6.4 を参照）。

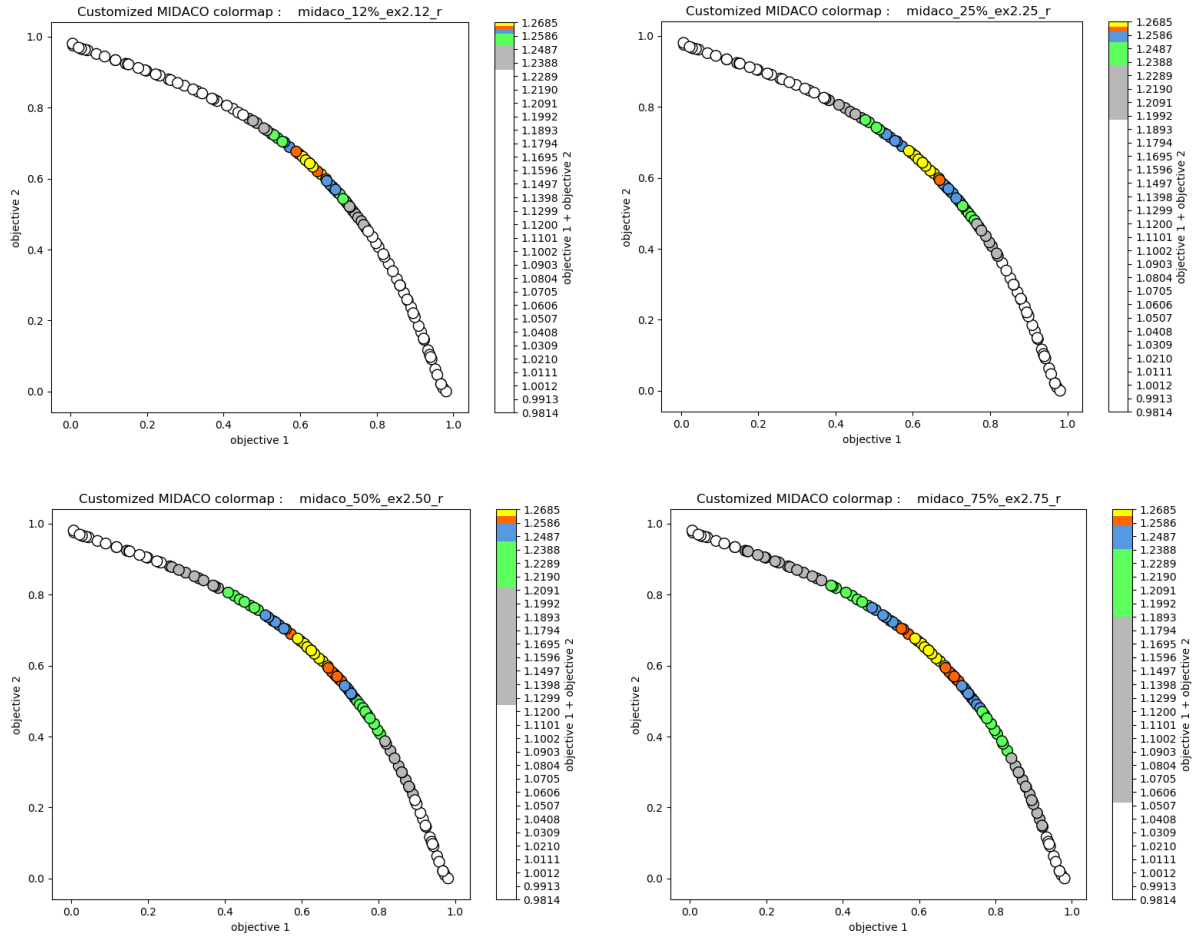


図4 MIDACO カラーマップをカスタマイズした Fonesca ベンチマークの４例

7 並列化

MIDACO には、複数の解の候補を並列評価する機能があります。進化的アルゴリズムの分野では、こうした機能が同時評価 *co-evaluation* や細粒度 *fine-grained* 並列化とも呼ばれています。下の図 5 は **P** 個の解の候補（x1, x2, x3, ... xP）の **ブロック**が並列評価に渡され、対応する目的関数値と制約関数値（[f1,g1],...,[fP,gP]）が MIDACO に返される様子を示しています。

最適化問題が CPU 時間のかかる問題、つまり目的関数と制約条件の 1 回の評価にかなりの時間を要する問題であれば、問題解決に要する時間全体を短縮する目的での並列化が、きわめて有益です。新しい研究（Schlueter & Munetomo [44]）では、並列化による MIDACO の潜在的な高速化が、200 のベンチマーク問題でほぼ直線的な向上 を示しており、最も効果的であることが数値的に実証されました。並列化係数 $P = 10$ の場合、潜在的な高速化は約 10 倍であり、並列化係数 $P = 100$ の場合、潜在的な高速化は約 70 倍にもなります ([44] の図 4 を参照)。ただし、並列化に伴うオーバーヘッドのため、このような高速化が期待できるのは、目的関数や制約条件の計算に **CPU 時間がかかる場合**のみとなります。この問題については、サブセクション7.2 でさらに詳しく説明します。

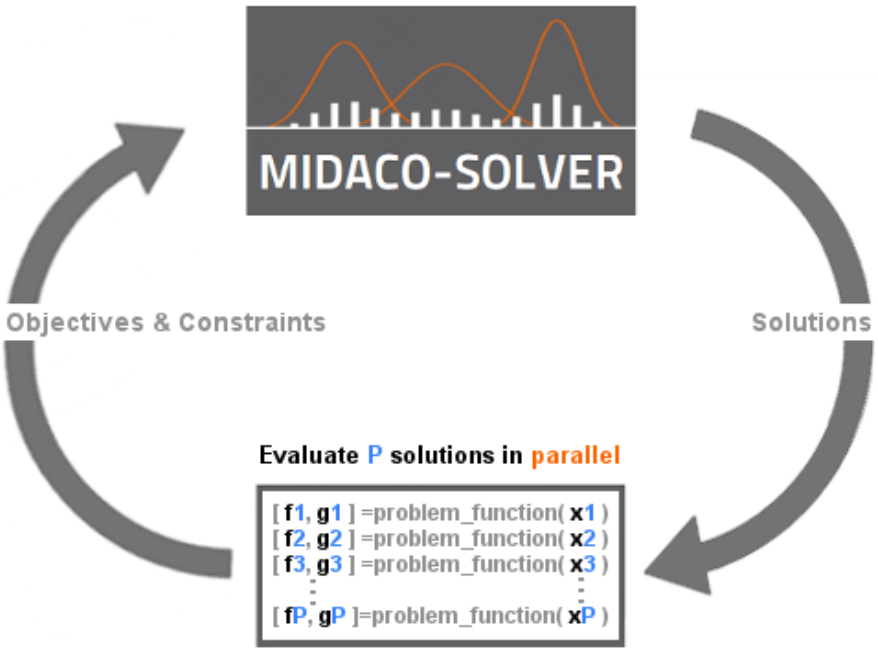


図5 MIDACO による **P 個**の解ブロック の**並列評価**

7.1 MIDACO による並列計算の実行

MIDACO の並列計算モードは、簡単に実行できます。様々な言語（Matlab、Python、C++、R、Java、C#、Fortran）に対応するサンプルが、オンラインで提供されています。

http://www.midaco-solver.com/index.php/more/parallelization

なお、MIDACO の並列化機能は、（openMP や MPI のような）言語や手法に限定されず 、GPGPU や Hadoop/Spark などの他の言語や並列化手法でも利用できます。MIDACO はリバースコミュニケーション reverse communication に基づいており、並列化機能は実質的にあらゆる言語／アプローチで可能です。

7.2 並列化のオーバーヘッド

並列計算では計算上ある程度のオーバーヘッドが生じるため、MIDACO の並列モードでの実行による効果は、主に 2 つの要因に左右されます。プログラミング言語／アプローチと、目的関数と制約条件を評価する実際の CPU 時間です。いくつかの言語（とくに Matlab）では、計算時のオーバーヘッドが非常に大きくなる可能性があるため、並列化しても、全体の最適化時間が短縮されず、かえって増加させてしまうことがあります。これは通常、問題関数の評価が、たとえば 1 回の評価に 0.00001 秒もかからないような、短時間である場合に起こります.

表 3は、プログラミング言語別に、並列化の効果が期待できる最小評価時間の大まかな目安を示しています。これらの時間は、実際の CPU のスペックや利用可能な並列スレッド数に左右されるため、実際にかかる時間には増減があります。特定のケースでの並列化が有益かどうか、ユーザーご自身が実験してみることをお勧めします。

表3: 並列化効果が期待できる最小評価時間

言語	最小評価時間
Matlab	1.0 秒
Python	0.01 秒
R	0.01 秒
Java	0.001 秒
C/C++	0.001 秒
Fortran	0.001 秒

8 Tips & Tricks

このセクションでは、よく見られる最適化の事例における上級者向けのヒントを説明します。

8.1 制約条件の取り扱い

制約条件の取り扱いは、最適化の中でも重要かつ難しい分野です。MIDACO は、最大で数百、時には数千の制約条件を持つ問題を解くことができます（たとえば、MIDACO ウェブサイトのベンチマークページや Schlueter and Munetomo[44] を参照)。MIDACO は等式制約だけでなく不等式制約にも対応しており、通常は不等式制約がより解きやすくなっています。MIDACO はブラックボックスのコンセプトに基づいており、制約条件の関数特性には縛られません。したがって、制約条件は線形でも非線形でもかまいませんし、微分可能でなくてもかまいません。

MIDACO を使って制約条件が多い、かつ／または難しい問題を効率的に解くためには、複数回の実行によるカスケードアプローチ *cascading* approach をお勧めします。多くの制約条件がある問題を解く上で最も重要なパラメータは、制約条件違反を測定する精度（PARAM(1), セクション4.1) です。多くの制約条件がある問題では、デフォルトの精度 0.001 では、ゼロから解を得るには難しすぎたり、時間がかかりすぎたりするかもしれません。したがって、最初の実行時に精度を上げる必要があります。最初の実行に適した精度の値は、事例ごとに異なり、たとえば 0.1、0.5、あるいは 1.0 かもしれません。このように PARAM(1) の値を高くすると、MIDACO は通常、実現可能な解をより早く見つけ、実現可能な領域が見つかれば、目的関数値の最小化をより速く進めることができます。

上のように最初に緩やかな制約条件精度で実行した後で、解の精度の絞り込みを開始できます。緩やかな制約条件精度で暗示された実行可能な解は、0.01、0.001、またはそれ以下のより正確な精度でさらに実行する際の開始点として利用できます。このような絞り込み実行のためには、FOCUS パラメータを有効にすることをお勧めします (セクション 4.6)。何回の微調整を行うか、ACCURACY と FOCUS をどのように設定するかは、それぞれの事例により異なります。下の表4 は、難しい制約付き問題を数回の実行で解決するための ACCURACY と FOCUS の設定に関する大まかな一例 を示しています。なお、表4では、最終的な解が 0.00001 以下の制約条件違反となることを前提としています。

表4: 複数回実行の設定例

実行	ACCURACY	FOCUS	開始点
1 回目	0.5	0.0 (デフォルト)	ゼロから
2 回目	0.1	-10.0	1 回目の解
3 回目	0.01	-100.0	2 回目の解
4 回目	0.001	-1000.0	3 回目の解
5 回目	0.00001	-100000.0	4 回目の解

表 4 の仮想的な複数回の実行設定では、制約付き問題を希望の制約条件精度 0.00001 の解にするために、合

計 5 回の実行が必要です。FOCUS パラメータには、「-」フラグが付いていることに注意してください。これは、上記の例では 2 回目から 5 回目までの実行が、絞り込み実行としてのみ考慮されるためです。

8.2 高度な非線形問題

制約条件が多い、かつ／または難しい問題を解く場合と同様に、高度な非線形問題（一般的に非常に解決が困難な問題）では、複数回実行を設定することが有効な場合があります。このような事例では、最初の実行は、適切な初期解を提供するためのゼロからの *from scratch* 実行の役目を果たし、その後の実行でさらに改善されます。こうした絞り込み実行のための重要なパラメータは FOCUS です（セクション4.6) 参照)。下の表5は、 2回目と 3回目の実行が絞り込み実行となる 3 回の実行の例を示しています。

表5: 複数回実行の設定例

実行	FOCUS	開始点
1回目	0.0 (デフォルト)	ゼロからの実行
2回目	100.0	1回目の解
3回目	-10000.0	2回目の解

表5において、 2回目の実行では FOCUS パラメータに「-」フラグを付けていない点にご注意ください。これは、 2回目の実行が絞り込み実行ながらも、MIDACO がさらなる領域を探索できるようにするためです。これとは対照的に、3 回目の実行では「-」フラグ付きの FOCUS を仮定しています。これは、3 回目の実行が絞り込みのための最終的な改善であることを意味しています。

8.3 大規模最適化問題

数百、数千の変数を持つ問題では、パフォーマンス向上のために一般的に、FOCUS、ANTS、KERNEL パラメータの調整が有効です。最初にゼロから実行する際には、FOCUS パラメータに 10 や 50、100 といった値が有効かもしれません。その理由は、大規模な問題では探索空間の縮小がより大きな影響を与えるため、小規模な問題よりも FOCUS パラメータの効果が大きくなる可能性があるからです。ANTS と KERNEL のパラメータは、[ANTS=2,KERNEL=2] 、[ANTS=5,KERNEL=20]、[ANTS=10,KERNEL=50] のように設定すると、パフォーマンスが向上するかもしれません。また、ANTS と KERNEL のパラメータの値を小さくすると、探索空間の探索が少なくなるため、収束が早くなります。

他方で、上記の設定例では、局所最適解への収束が最適ではない可能性があります。大規模な問題は一般的に解くのが難しいため、解の質よりも実行時間の短縮が重要な場合には、このような局所最適解も許容されるかもしれません。

8.4 CPU 時間のかかる事例

MIDACO を使って CPU 時間のかかるアプリケーションを解く場合、並列化を行うことでその性能を大幅に向上させることができます。ひとつの目的関数と制約条件の評価にコストがかかる（たとえば、数秒以上かかる）事例では、問題を解くための所要時間全体が、並列化によって平均して常に（大幅に）短縮されます。さらに、並列化のレベルは高ければ高いほどよいとされています。Schlueter and Munetomo [44] では、並列化

の度合いによって、MIDACO が必要とする逐次ステップ数 100 回が約 70 分の 1 になることが実証されました。CPU 時間のかかる事例では、100 スレッドのクラスタが利用できる場合、MIDACO には約 70 倍の実行時間が割り当てられ、2 ヶ月 (≈ 60 日間) かかるところが 1 日以下の時間で実行できることになります。

並列化および大規模最適化問題を扱う場合（セクション8.3を参照）と同じ推奨事項（ANTS、KERNEL、FOCUS の設定）が、CPU 時間のかかる事例にも該当しますは、適用できます。これは、大規模事例と CPU 時間のかかる事例の両方のシナリオにおいて、探索空間（すなわち、関数評価の数）の削減が大きな影響を与えるからです。すでに述べたように（セクション 8.3参照)、推奨される設定では、収束は速くなりますが、最適解が得られない可能性があります。とはいえ、CPU 時間のかかる事例では、探索に膨大な時間を要する大域的な最適解よりも、合理的な時間で到達できる最適ではないが良好な解の方が好ましいという場合には、それも許容できるかもしれません。

8.5 非線形方程式系を解く

MIDACO は、何百、何千もの制約条件を持つ問題を扱うことができるため、(通常、特定の目的関数が存在しない) 非線形方程式系を解くのに利用できます。その際、すべての制約条件を制約関数として定式化し、目的関数を空白（たとえば、定数）にしておく代わりに、最もむずかしい制約条件を目的関数として定式化することをお勧めします。このようにすると、MIDACO は大部分の制約条件を満たす解をより簡単に見つけることができ、後で絞り込み実行を行う際に（目的関数として与えられた）最も難しい制約条件を満たすことに集中できます。.

8.6 マルチモーダル最適化

マルチモーダル最適化では、単一の大域的最適解だけでなく、すべての局所最適解のセットを見つけることを目指します。MIDACO は進化的アルゴリズムに基づいているため、広大な探索空間を探索し、結果として（マルチモーダルな問題特性を持つ場合）多くの局所最適解に入ることになります。履歴ファイルの作成（セクション2.2参照）を有効にすれば、評価されたすべての解を追跡できるため、結果として、探索プロセス中に見つかったすべての局所最適解をさらに精査することができます。

8.7 開始点の複数登録

MIDACO を並列化して実行する場合、複数の開始点を登録することができます。このオプションは、非常に多くの CPU 時間を必要とする事例（1 回の評価に数時間を要するもの）で効果的です。MIDACO のサンプルテンプレートは、デフォルトで単一の開始点のみを登録しており、その開始点は、P 個の解ベクトル X を次々に格納する XXX 配列に複製されます。Matlab に対応するソースコードは、ゲートウェイ"midaco.m "の 100 行目付近にあります。

```
103  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
104  xxx = zeros(1,P*n);
105  for c = 1:P
106      for i = 1:n
107          xxx((c-1)*n+i) = x0(i);
108      end
109  end
110  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

複数のスタートポイントを XXX 配列に格納したい場合は、XXX fill up コマンドを修正して、ユーザーが手動で登録する必要があります。たとえば、P=3 で、3 つの異なる開始点がある場合、修正された"midaco.m " ゲートウェイは以下の通りです。

```
103  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
104  xxx = zeros(1,P*n);
105  % Fill up XXX array with 3 different starting points
106  for i = 1:n
107      xxx( 0*n + i ) = starting_point_1(i);
108      xxx( 1*n + i ) = starting_point_2(i);
109      xxx( 2*n + i ) = starting_point_3(i);
110  end
111  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

XXX 配列を埋めるためのソースコードのコマンドは、言語によって多少異なるとしても、基本的には常に同じ目的を果たしています。なお、開始点として異なるランダムな解を引き渡すことも可能で、それが効果的な場合もあります。

8.8 MIDACO を用いた並列化オーバークロック

評価時間が大幅に変化し（たとえば、数秒で終わる評価もあれば、数分または数時間かかる評価もある）、十分な数のスレッド／コアによる並列化を行えるような CPU 時間のかかるアプリケーションの場合、MIDACO の並列化係数をオーバークロック *overclock* することが効率的です。オーバークロックとは、実際に利用可能な物理スレッド/コア数よりも大きな並列化係数 P を割り当てることを意味します。このアプローチは、ハイパフォーマンス・コンピューティング HPCではオーバーサブスクリプションとも呼ばれています。並列化係数 P は自由に選ぶことができるため、実際に利用可能なスレッド／コア数よりも大きい (または小さい) 任意の整数値に設定できます。

たとえば、32 個の CPU(それぞれが 1 コア) のクラスターが、MIDACO を実行するマスターノードの関数評価として動作するとします (このような設定は、たとえば Spark で可能です)。MIDACO に P=64 または P=128 の並列化係数を割り当てると、実際に利用可能なコア数を超えることになりますが、全体的な処理の高速化につながる可能性があります。その理由は、高速な計算ソリューションを評価した後、P=32 の場合ではしばらくの間アイドル状態になってしまうコアを利用することで、クラスタ内のコアのプールをより効果的に使用できるからです。

注:1 台のマシンでもオーバークロックが有効な場合があります。ただし、すべての解の評価に同等の CPU タイムが必要な事例では、オーバークロックは推奨できません 。

9 IFLAG メッセージ

このセクションでは、MIDACO が**情報フラグ**として使用する IFLAG 値のリストについて説明します。MIDACO は、最終解の情報、警告、入力エラーを示すために、さまざまな IFLAG 値を報告します。最終解メッセージの場合には、1〜7 の IFLAG 値が記載され、計算終了の理由や解が実行可能かどうかを示します。新たな最適化問題を設定する際にはしばしば、いくつかの IFLAG エラーメッセージ（IFLAG=204 限界値のエラーなど）に遭遇します。これらは通常、簡単に修正できるもので、大きな問題ではありません。以下に、すべての IFLAG 値のリストを示します。MIDACO は負の IFLAG 値を内部通信にのみ使用することにご注意ください。

9.1 最終解メッセージ (IFLAG = 1 ∼ 9)

表 6 は、MIDACO が最終解とともに報告する IFLAG メッセージの説明です。

表6: IFLAG が示す MIDACO の最終解メッセージ

IFLAG	
1	実行可能解、MIDACO は MAXEVAL または MAXTIME により終了
2	実行不可能解、MIDACO は MAXEVAL または MAXTIME により終了
3	実行可能解、MIDACO は ALGOSTOP により自動終了
4	実行不可能解、MIDACO は ALGOSTOP により自動終了
5	実行可能解、MIDACO は EVALSTOP により自動終了
6	実行不可能解、MIDACO は EVALSTOP により自動終了
7	実行可能解、MIDACO は FSTOP により自動終了

9.2 警告メッセージ (IFLAG = 10 ∼ 99)

表 7 では、最適化計算プロセスの最初に警告として報告される IFLAG メッセージを説明します。警告は無視できますが、問題設定の不備を示す場合があります。

表7: IFLAG が示す MIDACO の警告メッセージ

IFLAG	
51	X(i) が +/- 10^{16} よりも大きい／小さい（極端な値は避けること）
52	XL(i) が +/- 10^{16} よりも大きい／小さい（極端な値は避けること）
53	XU(i) が +/- 10^{16} よりも大きい／小さい（極端な値は避けること）
71	XL(i) = XU(i) （変数が固定されている）
81	F(1) の開始点 X が非数 NaN 値である
82	G(X) の開始点 X が非数 NaN 値である
91	FSTOP が +/- 10^{16} よりも大きい／小さい
92	ORACLE が +/- 10^{16} よりも大きい／小さい

9.3 エラーメッセージ (IFLAG = 100 ∼ 999)

表8と表9 では、最適化を実行する際に最初にエラーとして報告される IFLAG メッセージを説明しています。エラーメッセージが表示された場合、MIDACO は最適化の実行を停止します。

表8: IFLAG が示す MIDACO のエラーメッセージ

IFLAG	メッセージの説明
100	P <= 0 または P > 10^{99} である
101	O <= 0 または O > 10^{9} である
102	N <= 0 または N > 10^{99} である
103	NI < 0 である
104	NI > N である
105	M < 0 または M > 10^{99} である
106	ME < 0 である
107	ME > M である
201	X(i) が NaN 値をもつ
202	XL(i) が NaN 値をもつ
203	XU(i) が NaN 値をもつ
204	X(i) < XL(i) である
205	X(i) > XU(i) である
206	XL(i) > XU(i) である
	注意：*PARAM(i)* の配列インデックスは、ゼロではなく 1 で始まります
301	PARAM(1) < 0 または PARAM(1) > 10^{99} である
302	PARAM(2) < 0 または PARAM(2) > 10^{99} である
303	PARAM(3) が +/- 10^{99} よりも大きい／小さい
304	PARAM(4) < 0 または PARAM(4) > 10^{99} である
305	PARAM(5) が +/- 10^{99} よりも大きい／小さい
306	PARAM(6) が離散値ではない、または PARAM(6) > 10^{99} である
307	PARAM(7) < 0 または PARAM(7) > 10^{99} である
308	PARAM(8) < 0 または PARAM(8) > 100 である
309	PARAM(7) < PARAM(8) である
310	PARAM(7) > 0 だが PARAM(8) = 0 (ANTS と KERNEL はともに利用すること)
311	PARAM(8) > 0 だが PARAM(7) = 0 (ANTS と KERNEL はともに利用すること)
312	PARAM(9) が +/- 10^{99} よりも大きい／小さい
321	PARAM(10) >= 10^{99} である
322	PARAM(10) が離散値ではない
331	PARAM(11) < 0 または > 0.5 である

表9: IFLAG が示す MIDACO のエラーメッセージ（続き）

IFLAG	メッセージの説明
344	パレートフロント (PF) ワークスペース **LPF が小さすぎる**。LPF のサイズは少なくとも (O+M+N)*PARETOMAX + 1、デフォルトでは PARETOMAX=1000 （セクション 4.10）
347	PARAM(5) > 0 だが PARAM(5) < 1 である
348	PARAM(5): オプション追加の EVALSTOP 精度 > 0.5 である
350	PARAM(12) < -1 または PARAM(12) > 1 だが離散値ではない
351	PARAM(13) < 0 または PARAM(13) > 3 である
352	PARAM(13) が離散値ではない
399	PARAM(i) が NaN 値をもつ
401	ISTOP < 0 または ISTOP > 1 である
402	開始点が all-different 条件を満たしていない
501	倍精度ワークスペースのサイズ **LRW が小さすぎる**。 RW 配列のサイズを大きくする。RW は以下のサイズ以上。 LRW = 120*N+20*M+20*O+20*P+P*(M+2*O)+O*O+5000
502	LRW 内部チェックエラー（サポートにご連絡ください）
601	整数ワークスペースのサイズ **LIW が小さすぎる**. IW 配列のサイズを大きくする。IW は以下のサイズ以上。 LIW = 3*N+P+1000
602	LIW 内部チェックエラー（サポートにご連絡ください）
701	入力チェックなし！MIDACO は最初に IFLAG = 0 の呼び出し必須
881	X の整数部分が連続した（非離散的な）値を含む
882	XL の整数部分が連続した（非離散的な）値を含む
883	XU の整数部分が連続した（非離散的な）値を含む
900	ライセンスキーが誤っているか破損している
999	N > 4 である。無料版は変数４つまで。

参考文献

[1] Abolhassani, A., .Harner, J., Jaridi, M., Gopalakrishnan, B.: *Productivity enhancement strategies in North American automotive industry.* Int. J. Prod. Res., 8(3), pp.1–18 (2017)

[2] Alghamdi W.Y., Wu H., Zheng W., Kanhere S.S.: *Constructing A Shortest Path Overhearing Tree With Maximum Lifetime In WSNs.* Hawaii International Conference on System Sciences (HICSS-49) (2016)

[3] Allugundu I., Puranik P., Lo Y.P. and Kumar A.: *Acceleration of distance-to-default with hardware-software co-design.* 22nd International Conference on Field Programmable Logic and Applications (FPL) (2012)

[4] Askin, T., Pornet, P.C., Vratny, M., Schmidt, M.: *Optimization of Commercial Aircraft Utilizing Battery based Voltaic-Joule/Brayton Propulsion.* Journal of Aircraft 54(1), pp. 246–261 (2016)

[5] Astos Solutions GmbH: *Low-Thrust Orbit Transfer Trajectory Optimization Software (LOTOS).* Stuttgart, Germany (2016)

[6] Bahbahani M.S., Baidas M.W., Alsusa E.A.: *A Distributed Political Coalition Formation Framework for Multi-Relay Selection in Cooperative Wireless Networks.* IEEE Transactions on Wireless Communications, Volume 14 , Issue: 12, pp. 6869 - 6882 (2016)

[7] Bahbahani M.S., Alsusa E.A.: *Relay Selection for Energy Harvesting Relay Networks using a Repeated Game.* IEEE Wireless Communications and Networking Conference (WCNC), At Doha, Qatar (2016)

[8] Baidas M.W. and MacKenzie A.B.: *On the Impact of Power Allocation on Coalition Formation in Cooperative Wireless Networks.* IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (2012)

[9] Baidas M.W. and Alsusa E.A.: *Power allocation, relay selection and energy cooperation strategies in energy harvesting cooperative wireless networks.* Wirel. Commun. Mob. Comput., DOI: 10.1002/wcm.2668 (2016)

[10] Baidas M.W. and Masud M.: *Energy-efficient partner selection in cooperative wireless networks: a matching-theoretic approach.* International Journal of Communication Systems, Volume 29, Issue 8, pp 1451-1470 (2016)

[11] Chagwiza G., Musekwa S., Jones B., Mtisi S.: *Impact of new water sources on the overall water network: an optimisation approach.* International Journal of Mathematics and Statistics Research, Vol.1, No.1, pp. 32-41 (2014)

[12] Chakraborty, D., Wang, T., Monika, A., Manfred, J., Christoph, L., Lambert, M., Schueler, W.: *Adapting Douglas-fir forestry in Central Europe: evaluation, application, and uncertainty analysis of a genetically based model.* European Journal of Forest Research, 135(5), pp. 919–936 (2016)

[13] Comas M.: *Application of sales forecasting for new products.* Technical report, Escola tecnica superior d'enginyeria industrial de Barcelona, Spain (2012)

[14] Dell I., Lekszyck T., Pawlikowski M., Grygoruk R., Greco L. : *Designing a light fabric metamaterial being highly macroscopically tough under directional extension: rst experimental evidence.* Zeitschrift

fur angewandte Mathematik und Physik (ZAMP) Vol 66 (6), pp. 3473-3498 (2015)

[15] Duquenne, B.: *Optimization tool dedicated to the validation process for the automotive industry.* MSc Thesis, University of Liege, Belgium (2017)

[16] Esche E.: *MINLP optimization under uncertainty of a mini plant for the oxidative coupling of methane.* PhD-Thesis, Fakultaet III, Prozesswissenschaften, Technical University of Berlin (2015)

[17] Faramondi, L., Oliva, G., Panzieri, S., Pascucci, F., Schlueter, M., Munetomo, M., Setola, R.: *Network Structural Vulnerability: A Multiobjective Attacker Perspective.* IEEE Transactions on Systems, Man, and Cybernetics: Systems, 99, pp.–1-14 (2018)

[18] European Space Agency (ESA) and Advanced Concepts Team (ACT): GTOP database - global optimisation trajectory problems and solutions. (2016)

[19] Grujic I., Nilsson R.: Model-based development and evaluation of control for complex multi-domain systems: attitude control for a quadrotor UAV. Technical report ECE-TR-23, Aarhus University, Denmark (2016)

[20] Haenel M., Kuhn S., Henrich D., Gruene L. and Pannek J.: *Optimal camera placement to measure distances regarding static and dynamic obstacles.* Int. J. of Sensor Networks, 12(1), pp.25–36 (2012)

[21] Hunter, J. D.: *Matplotlib: A 2D graphics environment.* Computing In Science & Engineering, 9(3), pp.90–95 (2007)

[22] Kahar, N.H.B.A., Zobaa, A.F. : *Optimal single tuned damped filter for mitigating harmonics using MIDACO.* IEEE Industrial and Commercial Power Systems Europe, DOI: 10.1109/EEEIC.2017.7977541, (2017)

[23] Kahar, N.H.B.A., Zobaa, A.F. : *Application of mixed integer distributed ant colony optimization to the design of undamped single-tuned passive filters based harmonics mitigation.* Swarm and Evolutionary Computation, https://doi.org/10.1016/j.swevo.2018.03.004, *in press* (2018)

[24] Lou X.: *Acceleration of Distance-to-Default with GPU.* Master-Thesis, School of Information & Communication Technology Royal Institute of Technology Stockholm, Sweden (2012)

[25] Mahajan N.R., Mysore S.P.: *Combinatorial neural inhibition for stimulus selection across space.* biorxiv, doi.org/10.1101/243279 (2018)

[26] Minguijon Pallas, P.: *Cubesat Deployment Trajectories for the Asteroid Impact Mission.* MSc Thesis, Delft University of Technology, Netherlands (2017)

[27] Mohammed, S. M.: *Exergoeconomic analysis and optimization of combined cycle power plants with complex configuration.* PhD Thesis, Univ. of Belgrade, Fac. Mech. Eng., Serbia (2015)

[28] Mukalu, M.S., Lijun, Z., Xiaohua, X.: *A Comparative Study on the Cost-effective Belt Conveyors for Bulk Material Handling.* Energy Procedia, Volume 142, pp. 2754–2760 (2017)

[29] Nie C., Wu H., Zheng W.: *Lifetime-Aware Data Collection Using A Mobile Sink in WSNs with Unreachable Regions.* MSWiM17, November 21-25, 2017, Miami, FL, USA 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems Pages 143-152, DOI:10.1145/3127540.3127544 (2017)

[30] Perez R.E., Jansen P.W.: *Effect of Passenger Preferences on the Integrated Design and Optimization of Aircraft Families and Air Transport Network.* 17th AIAA Aviation Technology, Integration, and Operations Conference, DOI: 10.2514/6.2016-3748 (2017)

[31] Redutskiy Y.: *Oilfield development and operations planning under geophysical uncertainty.* Engineering Management in Production and Services, Volume 9, Issue 3, Pages 10-27, doi.org/10.1515/emj-2017-0022 (2018)

[32] Rehberg M., Ritter J.B, Genzela Y., Flockerzi D. and Reichl U.: *The relation between growth phases, cell volume changes and metabolism of adherent cells during cultivation.* J. Biotechnol., 164(4), pp. 489–499 (2013)

[33] Schittkowski K.: *NLPQLP - A Fortran Implementation of a Sequential Quadratic Programming Algorithm with distributed and non-monotone Line Search (User Manual)*, Report, Department of Computer Science, University of Bayreuth (2009)

[34] Schlueter M., Egea J.A. and Banga J.R.: *Extended Ant Colony Optimization for non-convex Mixed Integer Nonlinear Programming*, Comput. Oper. Res. 36(7), pp. 2217–2229 (2009)

[35] Schlueter M., Egea J.A., Antelo L.T., Alonso A.A. and Banga J.R.: *An extended Ant Colony Optimization algorithm for integrated Process and Control System Design*, Ind. Eng. Chem. 48(14), pp. 6723–6738 (2009)

[36] Schlueter M. and Gerdts M.: *The Oracle Penalty Method*, J. Global Optim. 47(2), pp. 293–325 (2010)

[37] Schlueter M., Rueckmann J.J and Gerdts M.: *A Numerical Study of MIDACO on 100 MINLP Benchmarks*, Optimization, 61(7), pp. 873–900 (2012)

[38] Schlueter M.: *Nonlinear mixed integer based Optimisation Technique for Space Applications*, Ph.D. Thesis, School of Mathematics, University of Birmingham (UK) (2012)

[39] Schlueter M., Erb S., Gerdts M., Kemble S. and Rueckmann J.J.: *MIDACO on MINLP Space Applications*, Optimization, 51(7), pp.1116–1131 (2013)

[40] Schlueter M. and Munetomo M.: *Parallelization Strategies for Evolutionary Algorithms for MINLP*, Proc. Congress on Evolutionary Computation (IEEE-CEC), pp.635-641 (2013)

[41] Schlueter M. and Munetomo M.: *Parallelization for Space Trajectory Optimization*, Proc. World Congress on Computational Intelligence (IEEE-WCCI), pp. 832 - 839 (2014)

[42] Schlueter M.: *MIDACO Software Performance on Interplanetary Trajectory Benchmarks*, Advances in Space Research (Elsevier), Vol 54, Issue 4, Pages 744 - 754 (2014)

[43] Schlueter M., Yam C.H., Watanabe T., Oyama A.: *Parallelization Impact on Many-Objective Optimization for Space Trajectory Design*, *Int. J. of Machine Learning and Computing 6.1: 9-14.* (2016)

[44] Schlueter M. and Munetomo M.: *Numerical Assessment of the Parallelization Scalability on 200 MINLP Benchmarks*, *Proc. IEEE-WCCI, Vancouver, Canada* (2016)

[45] Takano A.T. and Marchand B.G.: *Optimal Constellation Design for Space Based Situational Awareness Applications* AAS/AIAA Astrodynamics Specialists Conference (Paper No. AAS11-543) (2011)

[46] Teichgraeber, H., Brodrick, P., Brandt, A.: *Optimal design and operations of a flexible oxyfuel natural gas plant.* Energy 141, DOI: 10.1016/j.energy.2017.09.087 (2017)

[47] Tilly, J., Niedermayer, K.: *Employment and Welfare Effects of Short-Time Work.* German Economic Association, Annual Conference: Demographic Change, Augsburg (2016)

[48] Ukritchon B., Boonyatee T.: *Soil Parameter Optimization of the NGI-ADP Constitutive Model for*

Bangkok Soft Clay. Geo. Eng. J. SEAGS & AGSSEA, Vol. 46(1), pp. 28-36 (2015)

[49] Wang K., Mao Y., Chen J., Yu S.: *The optimal research and development portfolio of low-carbon energy technologies: A study of China.* J. Clean. Prod., Vol 176, pp. 1065–1077 (2018)

[50] Weiss L., Koeke H., Schlueter M., Huehne C.: *Structural optimisation of a composite aircraft frame for a characteristic response curve*. Proc. Euro. Conf. Comp. Mat. (ECCM17), (2016)

[51] Wong S.I.: *On Lightweight Design of Submarine Pressure Hulls.* MSc Thesis, Delft University of Technology, Netherlands (2012)

[52] Zhao Q., Neveux T., Jaubert J.N., Mecheri M., Privat R.: *Design of SC-CO2 Brayton cycles using MINLP optimization within a commercial simulator.* 6th Inter. Supercritical CO2 Power Cycles Symposium, March 27-29, 2018, Pittsburgh, USA (2018)

[53] Zarko D., Kovacic M., Stipetic S., Vuljaj D.: *Optimization of electric drives for traction applications.* 19th Int. Conf. on Elec. Drives and Power Electronics (EDPE), Dubrovnik, pp. 15–32 (2017)